NASA/CR-1999-209119

Aviation System Analysis Capability Executive Assistant Development

Eileen Roberts, James A. Villani, Kevin Anderson, and Paul Book
Logistics Management Institute, McLean, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NAS2-14361

March 1999

Abstract

In this technical document, we describe the development of the Aviation System Analysis Capability (ASAC) Executive Assistant (EA) Proof of Concept (POC) and Beta version. We describe the genesis and role of the ASAC system, discuss the objectives of the ASAC system, provide an overview of components and models in the ASAC system, and describe the design process and the results of the ASAC EA POC and Beta system development. We also describe the evaluation process and results for applicable commercial off-the-shelf software. The document has seven chapters, a bibliography, and two appendices.

Contents

FIGURES

TABLES

Chapter 1 Introduction

National Aeronautics and Space Administration's Role in Promoting Aviation Technology

The United States has long been the world's leader in aviation technology for both civil and military aircraft. During the past several decades, U.S. firms have transformed this position of technological leadership into a thriving industry with large domestic and international sales of aircraft and related products.

Despite the industry's historic record of success, the difficult business environment of the past several years has stimulated concerns about whether the U.S. aeronautics industry will maintain its worldwide leadership position. Increased competition, both technological and financial, from European and other non-U.S. aircraft manufacturers has reduced the global market share of U.S. producers of large civil transport aircraft and cut the number of U.S. airframe manufacturers to only one. Order cancellations and stretch-outs of deliveries by airlines, forthcoming noise abatement requirements, and environmental concerns create additional challenges for U.S. producers and purchasers of aircraft.

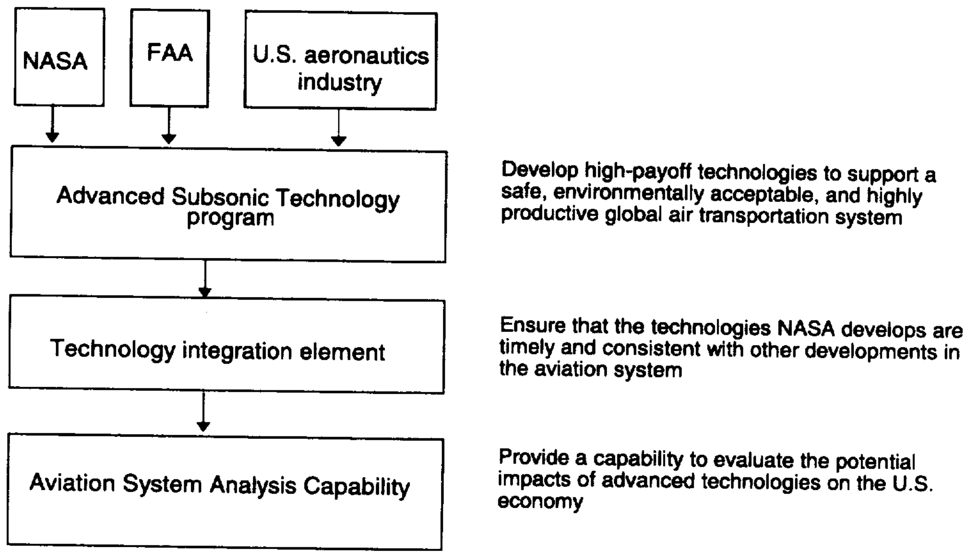
The primary role of the National Aeronautics and Space Administration (NASA) in supporting civil aviation is to develop technologies that improve the overall performance of the integrated air transportation system, making air travel safer and more efficient, as well as contributing to the economic welfare of the United States. NASA conducts much of the basic and early applied research that creates the advanced technology introduced into the air transportation system. Through its technology research program, NASA aims to maintain and improve the leadership role in aviation technology and air transportation held by the United States for the last half century.

The principal NASA program supporting subsonic transportation is the Advanced Subsonic Technology (AST) program, managed by the Subsonic Transportation Division, Office of Aeronautics, NASA Headquarters. In cooperation with the Federal Aviation Administration (FAA) and the U.S. aeronautics industry, the AST program develops high-payoff technologies that support the development of a safe, environmentally acceptable, and highly productive global air transportation system. NASA measures the long-term success of its AST program by how well it contributes to an increased market share for U.S. civil aircraft and aircraft component producers and the increased effectiveness and capacity of the national air transportation system.

NASA's Research Objective

To meet its objective of assisting the U.S. aviation industry with the technological challenges of the future, NASA must identify research areas that have the greatest potential for improving the operation of the air transportation system. Therefore, NASA seeks to develop the ability to evaluate the potential impact of various advanced technologies. By thoroughly understanding the economic impact of advanced aviation technologies and by evaluating the use of new technologies in the integrated aviation system, NASA aims to balance its aeronautical research program and help speed the introduction of high-leverage technologies. Figure 1-1 illustrates NASA's research objective.

Figure 1-1. NASA's Research Objective

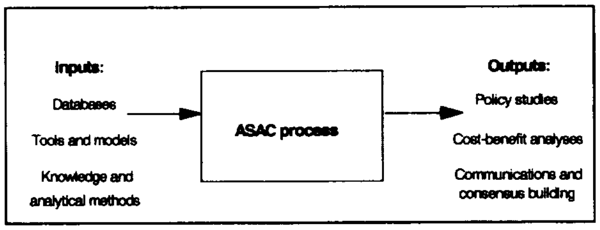

NASA
FAA
U.S. aeronautics industry
Advanced Subsonic Technology program
Develop high-payoff technologies to support a safe, environmentally acceptable, and highly productive global air transportation system
Technology integration element
Ensure that the technologies NASA develops are timely and consistent with other developments in the aviation system
Aviation System Analysis Capability
Provide a capability to evaluate the potential impacts of advanced technologies on the U.S. economy

Genesis of the Aviation System Analysis Capability

Technology integration is the element of the AST program designed to ensure that the technologies NASA develops are timely and consistent with other developments in the aviation system. Developing an Aviation System Analysis Capability (ASAC) is one of the objectives of the technology integration element. With this analytical capability, NASA and other organizations in the aviation community can better evaluate the potential economic impacts of advanced technologies.

ASAC is envisioned primarily as a process for understanding and evaluating the impact of advanced aviation technologies on the U.S. economy. ASAC consists of a diverse collection of models, databases, analysts, and individuals from the public and private sectors brought together to work on issues of common interest to organizations within the aviation community. ASAC will also be a resource available to those same organizations to perform analyses; provide information; and assist scientists, engineers, analysts, and program managers in their daily work. ASAC will provide this assistance through information system resources, models, and analytical expertise, and conducting and organizing large-scale studies of the aviation system and advanced technologies. Figure 1-2 displays this concept.

Figure 1-2. ASAC Process



Goals of the ASAC Project: Identify and Evaluate Promising Technologies

Developing credible evaluations of the economic and technological impact of advanced aviation technologies on the integrated aviation system is the principal objective of ASAC. These evaluations will then be used to help NASA program managers select the most beneficial mix of technologies for NASA investment, both in broad areas, such as propulsion or navigation systems, and in more specific projects within the broader categories. Generally, engineering analyses of this kind require multidisciplinary expertise, use several models of different components and technologies, and consider multiple economic outcomes and technological alternatives. These types of analyses are most effective if they include information and inputs from organizations and analysts from different parts of the aviation community. In this way, the studies incorporate the expertise of people around the United States and build acceptance from the start of the research effort.

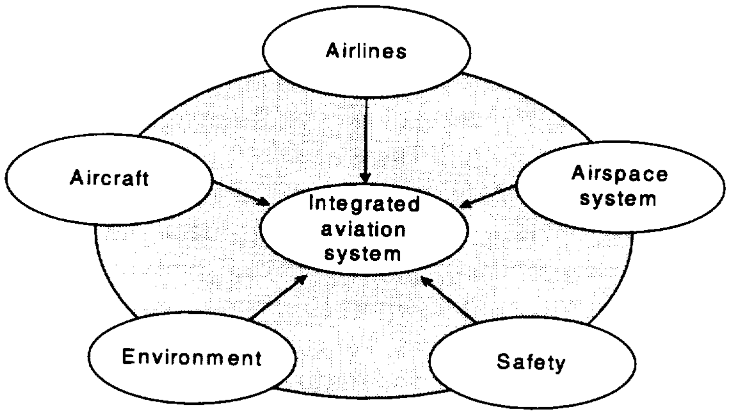
In addition to identifying broad directions for investments in technology, the program must also help researchers at NASA and elsewhere evaluate the economic potential of alternative technologies and systems. By better informing engineers about potential markets for technologies and data on how the current system works, ASAC will help NASA engineers incorporate their customers' needs more easily into their routine work. These types of problems most likely involve investigating technical designs for specific aircraft or subsystems that can readily replace existing equipment without requiring significant changes to other aviation components. With such information, researchers could more easily evaluate the utility of alternative designs and quickly estimate the value of their design concepts. Analysts from industry, government, and universities would also use ASAC in this way.

Approach to Analyzing the Integrated Aviation System

The most useful aviation technologies are not necessarily the most technically advanced. Rather, NASA and industry must invest in the technologies that have the most promising payoffs—those that clearly demonstrate a capacity for economically viable performance enhancements—from the perspective of those organizations that will purchase and operate the technologies.

Because new aviation technologies are introduced into a complex system, the potential impact of any proposed technology must be analyzed from a system-wide perspective. Otherwise, the potential impact may be overestimated or underestimated because of the unexamined interdependencies with other elements of the aviation system. Figure 1-3 shows the components of the integrated aviation system.

Figure 1-3. Components of the Integrated Aviation System



In summary, with the ASAC, users can develop credible evaluations of the economic and technological impact of advanced aviation technologies on all components of the integrated aviation system.

Document Overview

This technical document describes the system development of the ASAC Executive Assistant (EA). The document builds upon the work presented in the NASA Contractor Reports #201681, *ASAC Executive Assistant Architecture Description Summary*, Eileen Roberts and James A. Villani, April 1997, and #207679, *Aviation System Analysis Capability Executive Assistant Design*, Eileen Roberts and James Villani, et. al., May 1998, and it is composed of the following chapters:

- Chapter 1—Introduction
- Chapter 2—Components of the Aviation System Analysis Capability

- Chapter 3—ASAC Analyses
- Chapter 4—Design and Development Methodology
- Chapter 5—ASAC EA Proof of Concept
- Chapter 6—ASAC EA Beta Version
- Chapter 7—Conclusion.

In Chapters 1 through 3, the genesis and role of the ASAC system is described. We discuss the objectives of the ASAC system and provide an overview of components and models within the ASAC system.

The Design and Development Methodology chapter discusses the Domain-Specific Software Architecture (DSSA), and the DSSA approach to developing a system design. The chapter also describes the design tools used for the ASAC EA system.

The next two chapters, ASAC EA Proof of Concept and ASAC EA Beta Version, describe the requirements and goals of the ASAC EA system and includes the ASAC EA system design. The chapters also describe the development environment and process, the verification, and the testing. We address:

- DSSA Stage 4—Develop Domain Models
- DSSA Stage 5—Identify Reusable Artifacts.

DSSA stages 1 through 3 and part of stage 4 are detailed in the documents referenced above.

This document has a bibliography and two appendices:

- Appendix A—ASAC EA POC As-Run Test Procedures
- Appendix B—Abbreviations.

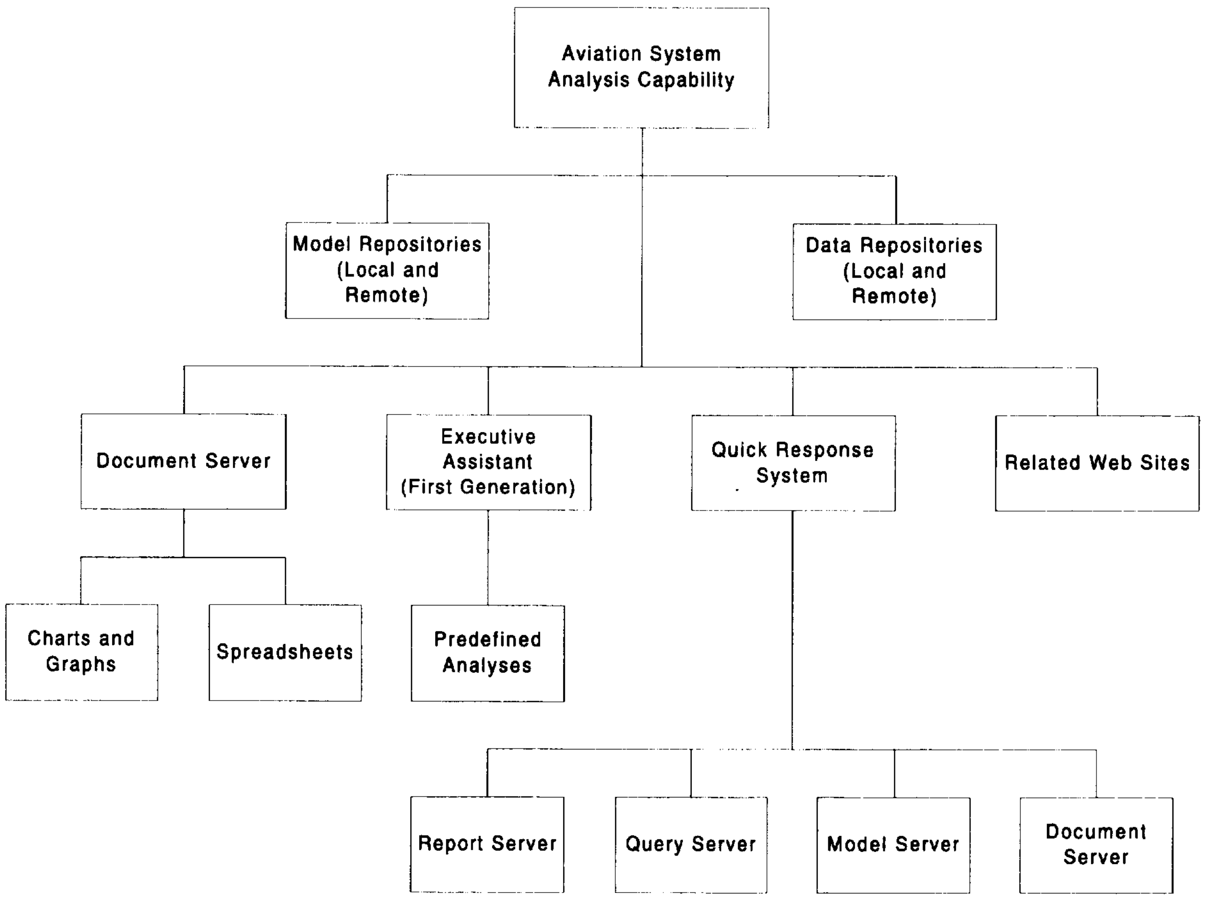
Chapter 2
Components of the ASAC

OVERVIEW

ASAC is a diverse collection of models, databases, analysts, and individuals from the public and private sectors brought together to work on the issues of common interest to organizations within the aviation community.

Figure 2-1 shows the major system components of ASAC.

Figure Chapter 2 -1. ASAC System Components



Most ASAC system components exist; others are under development. Two ASAC components, Document Server and the Related Web Sites are available to the general public. All other ASAC components are available on a restricted basis.

Information about the ASAC Executive Assistant (First Generation) can be found in *Aviation System Analysis Capability Executive Assistant Design*, referenced in Chapter 1. Information about the Quick Response System (QRS) and other ASAC components can be found in the NASA Contractor Report #201680, *Aviation System Analysis Capability Quick Response System Report for Fiscal Year 1997*, Eileen Roberts, James A. Villani and Paul Ritter, March 1998.

ASAC EXECUTIVE ASSISTANT

With the ASAC EA, researchers at NASA and elsewhere can quickly evaluate the economic potential of alternative technologies and systems. By providing inputs to and linking the many models and data that the ASAC system will comprise, the EA will provide an intelligent interface with which the user can perform detailed analyses. The ASAC EA Proof of Concept (POC) and Beta version development is the focus of this document.

Table Chapter 2 -1 outlines the proposed development schedule for the EA.

Table Chapter 2 -1. Proposed Development Schedule for the ASAC EA

Item	Year	Status
Define ASAC EA requirements	1995	Complete
Define the ASAC EA	1996	Complete
Develop the ASAC EA architecture	1996	Complete
Develop the Model Integration Prototype (First Generation ASAC)	1996–1997	Complete
Design and develop the ASAC EA Proof of Concept	1997–1998	Complete
Design, develop, and deploy the ASAC EA Beta version	1998	Ongoing
Design, develop, and deploy the ASAC EA version 1.0	1999	—
Refine the ASAC EA	1999	—

Chapter 3
ASAC Analyses

ASAC Models

The ASAC Model Integration Prototype (First Generation ASAC) was fielded in March 1997. It demonstrated the integration of six First Generation ASAC models, and was the first step in providing a robust, fully functional, ASAC EA. NASA and others used the ASAC Model Integration Prototype (First Generation ASAC) to perform selected economic analysis of aircraft technology and air traffic management improvements.

Additional models have been added to the ASAC Model Integration Prototype (First Generation ASAC) since its debut. It currently comprises a subset of the complete ASAC model network.

The ASAC Model Integration Prototype (First Generation ASAC) is available to authorized ASAC users (password protected). Users employ a World Wide Web (WWW) browser to access the system.

At present, seven models plus variants of two of the models, are in the ASAC Model Repositories. The models are listed in Table 3-1. New models will be added to the repositories as they are developed.

Table Chapter 3 -1. Contents of ASAC Model Repositories

Model	Operating System	Comment
ASAC Air Carrier Investment Model	HP-UX	Available via a WWW interface
ASAC Air Carrier Network Cost Model	HP-UX	Available via a WWW interface
ASAC Airport Capacity Model—Atlanta, Dallas-Fort Worth, Detroit, Los Angeles, New York La Guardia	HP-UX	Available via a WWW interface
ASAC Airport Delay Model— Atlanta, Dallas-Fort Worth, Detroit, Los Angeles, New York La Guardia	HP-UX	Available via a WWW interface
ASAC Flight Segment Cost Model (Cost Translator)	HP-UX	Available via a WWW interface
ASAC Flight Segment Cost Model (Mission Generator)	HP-UX	Available via a WWW interface
ASAC Noise Impact Model	Windows NT	Available via a WWW interface

SCHEMATIC OF ASAC MODELS

ASAC models are grouped into the following three analytical areas:

- 1.0 Aircraft and System Technologies
- 2.0 FAA Air Traffic Management
- 3.0 Environment.

Each model has a unique number. The number designates the model's analytical area, e.g., all model numbers that begin with a *2* belong to the FAA Air Traffic Management (ATM) analytical area. The models outlined in bold are available in the First Generation ASAC.

ASAC models can be combined to form analyses. For example, an analysis might comprise the following models:

2.1 ASAC Airport Capacity Model →

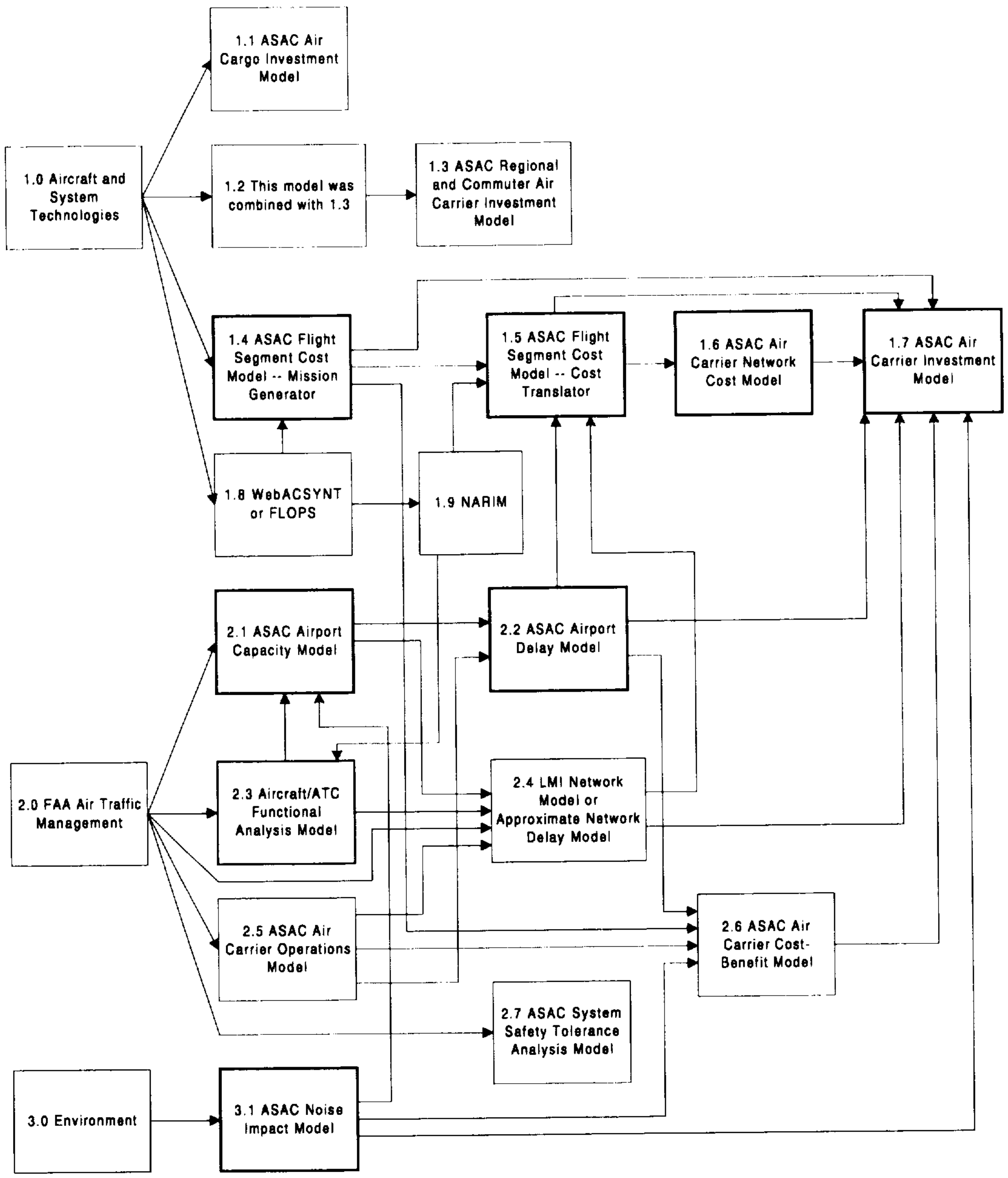
2.2 ASAC Airport Delay Model →

1.5 ASAC Flight Segment Cost Model—Cost Translator →

1.7 ASAC Air Carrier Investment Model.

Model links for each of the three analytical areas are shown in Figures 3-1.

Figure Chapter 3 -1. ASAC Model Links


1.1 ASAC Air Cargo Investment Model
1.0 Aircraft and System Technologies
1.2 This model was combined with 1.3
1.3 ASAC Regional and Commuter Air Carrier Investment Model
1.4 ASAC Flight Segment Cost Model -- Mission Generator
1.5 ASAC Flight Segment Cost Model -- Cost Translator
1.6 ASAC Air Carrier Network Cost Model
1.7 ASAC Air Carrier Investment Model
1.8 WebACSYNT or FLOPS
1.9 NARIM
2.1 ASAC Airport Capacity Model
2.2 ASAC Airport Delay Model
2.0 FAA Air Traffic Management
2.3 Aircraft/ATC Functional Analysis Model
2.4 LMI Network Model or Approximate Network Delay Model
2.5 ASAC Air Carrier Operations Model
2.6 ASAC Air Carrier Cost-Benefit Model
2.7 ASAC System Safety Tolerance Analysis Model
3.0 Environment
3.1 ASAC Noise Impact Model

Analyses Using ASAC Models

The above represented models can be used either alone or in combination to analyze specific AST program elements. A future NASA contractor report, *Aviation System Analysis Capability Executive Assistant Analyses*, will describe specific analyses that may be incorporated into the ASAC EA system.

Chapter 4
Design and Development Methodology

As discussed in the NASA Contractor Reports #201681, *ASAC Executive Assistant Architecture Description Summary,* and #207679, *Aviation System Analysis Capability Executive Assistant Design*, the DSSA is being used as a methodology for the ASAC EA system.

THE DSSA APPROACH

A domain engineering process is used to generate a DSSA. The goal of the process is to map user needs into system and software requirements that, based on a set of implementation constraints, eventually define a DSSA.

There are five stages in the DSSA domain engineering process. Each stage is further divided into steps or substages. The process is concurrent, recursive, and iterative. Therefore, completion requires several passes through each stage. The five stages in the domain engineering process are described in Table 4-1.

Table Chapter 4 -1. DSSA Stages

Stage	Title	Description	ASAC EA phase
1	Define the scope of the domain	Definition of what can be accomplished with emphasis on user needs	Architecture
2	Define/refine domain-specific elements	Similar to requirements analysis with emphasis on the problem space	Architecture
3	Define/refine domain-specific design and implementation constraints	Similar to requirements analysis with emphasis on the solution space	Architecture
4	Develop domain models and architectures	Similar to high-level design with emphasis on defining module and model interfaces and semantics	Architecture and design
5	Produce and gather reusable work products	Implementation and collection of reusable artifacts such as code and documentation	Design and development

DSSA stages 1, 2, 3, 4, and 5 (partial) were defined in the *ASAC Executive Assistant Architecture Description Summary* and *Aviation System Analysis Capability Executive Assistant Design.* An iteration of DSSA stage 4 and the remainder of DSSA stage 5 are addressed in this document.

DSSA Design Tools

Unified Modeling Language

Object-oriented design (OOD) is a development approach based on the organization of entities that have structure and behavior. It promotes the construction of well-defined systems and facilitates reuse and ease of modification. The Object Modeling Technique (OMT), used to develop the *ASAC Executive Assistant Architecture Description Summary*, was one method used to cover the system development process from the conceptualization phase through implementation. The author of OMT has collaborated with the authors of other OOD methodologies, namely Booch and Jacobson, to create the Unified Modeling Language (UML). UML is the successor to the past object-oriented design notations and has been proposed as a standard to the Object Management Group (OMG). UML notation is used to document the design. The Rational Rose visual modeling tool is used to automate this process.

A brief description of UML diagrams is found in Table 4-2.

Table Chapter 4 -2. Unified Modeling Language Diagram Definitions

Diagram	Description
Use case	A snapshot of one aspect of a system. The sum of all use cases is the external picture of a system.
Sequence	An interaction diagram that models message passing behavior between objects.
Collaboration	An interaction diagram that models message passing behavior between objects.
Package	Shows a high-level picture of components (packaged classes) and the dependencies among them.
Class	A description of the classes in a system and the interrelationships among them.
State	Shows all possible states for an object and how the object's state changes as a result of events.
Deployment	Shows the physical relationships among software and hardware components in the delivered system.
Activity	Is a flow chart of tasks or methods on a class.
Data flow	A depiction of the relationships among functions, usually within the problem domain.

The first seven diagrams are used to represent the ASAC EA design in this document. The other two diagrams may be used in the future (they are not required at this point).

The methodology associated with the UML notation is called Objectory, and is still being developed. Like UML notation, Objectory brings together the best aspects of the OMT, Booch, and OOSE (Jacobson) methodologies.

Class-Responsibility-Collaboration Card Technique

A technique called Class-Responsibility-Collaboration (CRC) Card is used to define the classes and class collaborations. CRC Card technique facilitates the process of discovering the real-world objects that make up a system and its public interfaces.

CRC cards are index cards that record

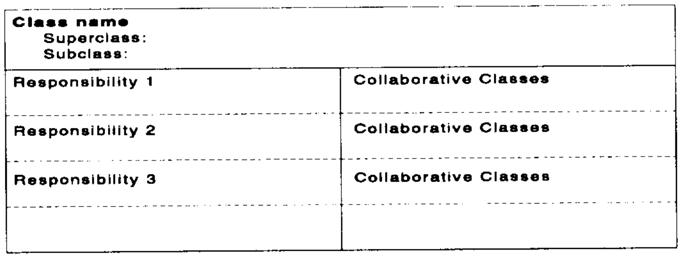
- ◆ suggested classes,
- ◆ their responsibilities,
  - ➢ what the classes know about themselves (knowledge responsibility)
  - ➢ what the classes do (behavior responsibility), and
- ◆ their relationship to other classes (collaboration).

CRC cards can optionally record

- ◆ class definitions and
- ◆ class attributes.

The front and optional back views of a CRC card are shown in Figures 4-1 and 4-2, respectively.

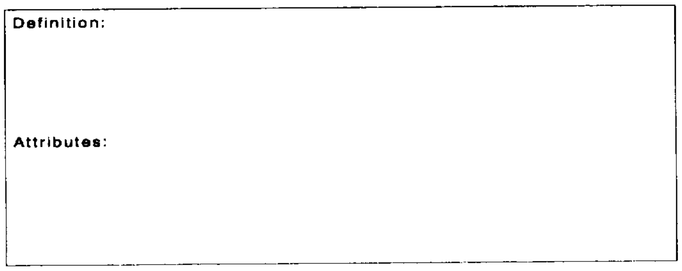
Figure Chapter 4 -1. CRC Card—Front View



Class name
Superclass:
Subclass:

Responsibility 1	Collaborative Classes
Responsibility 2	Collaborative Classes
Responsibility 3	Collaborative Classes

Figure Chapter 4 -2. CRC Card—Back View



Definition:

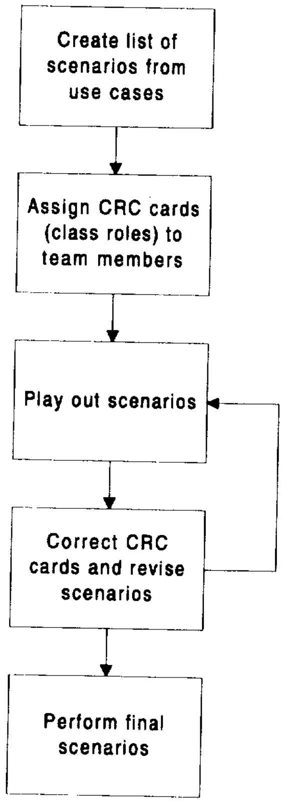
Attributes:

The CRC cards are used to role-play system scenarios. A person represents a class and responds to a request from another class based upon what is written on his or her CRC card. The role-play enables one to

- validate classes,
- ensure the identification of what the class knows and what the class does, and
- ensure all class hierarchies are identified.

The CRC card process is depicted in Figure Chapter 4 -3.

Figure Chapter 4 -3. CRC Card Process


Create list of scenarios from use cases
Assign CRC cards (class roles) to team members
Play out scenarios
Correct CRC cards and revise scenarios
Perform final scenarios

Design Patterns

Design patterns record experience in designing object-oriented software by naming, explaining, and evaluating important and recurring designs in object-oriented systems. An example of a design pattern is the Observer pattern, defined by Gamma, et al., as "a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically." The Observer, Flyweight, and Strategy design patterns were used in developing the ASAC EA design.

FURTHER READING

For more information about UML CRC cards and design patterns, see the following references:

[1] Fowler, Martin and Kendall Scott, "UML Distilled—Applying the Standard Object Modeling Language," Addison-Wesley, 1997.

[2] Rational Software Corporation UML Resource Center, "UML Document Set Version 1.1," September 1997, http://www.rational.com/uml/references/.

[3] Bellin, David and Susan Suchman Simone, "The CRC Card Book," Addison-Wesley, 1997.

[4] Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides, "Design Patterns—Elements of Reusable Object-Oriented Software," Addison-Wesley, 1995.

Chapter 5
ASAC EA Proof Of Concept

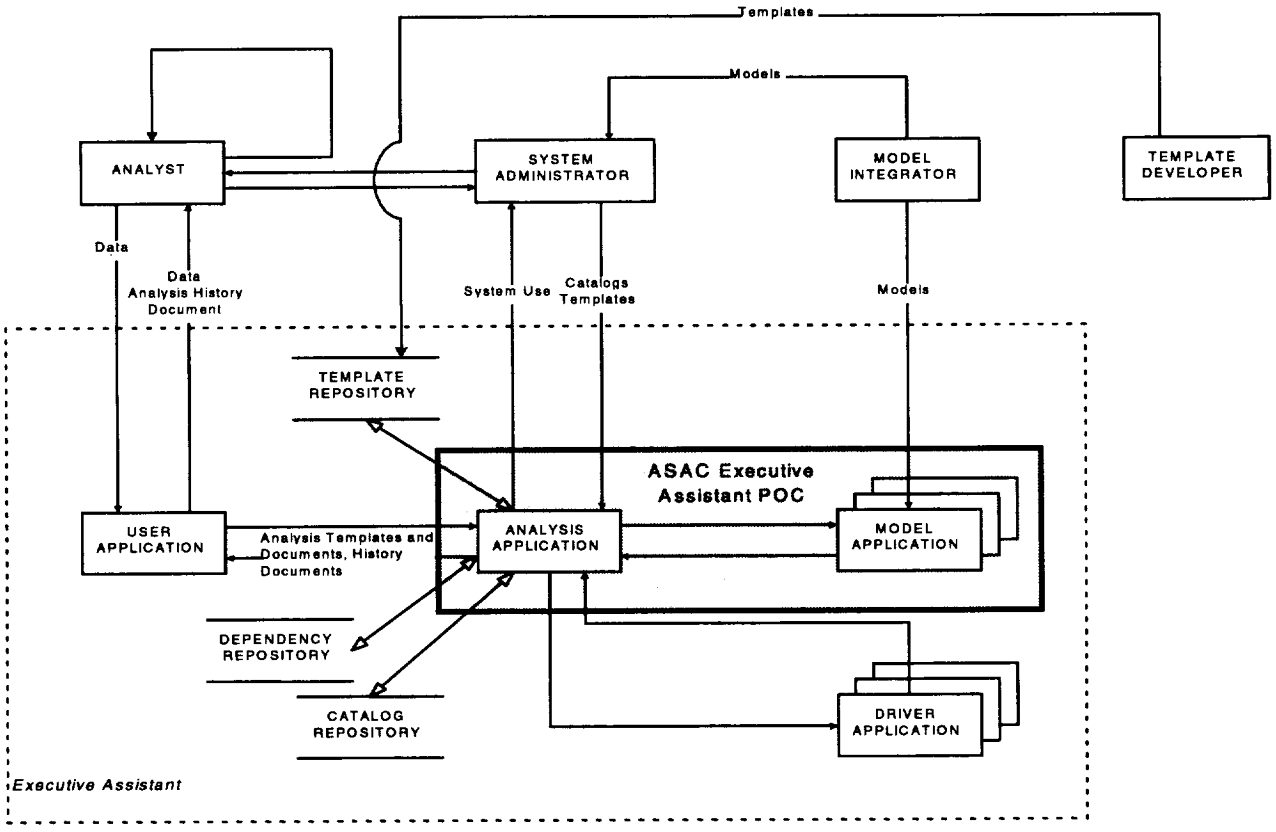
The DSSA approach was tailored to meet the needs of the ASAC development effort. The next two sections discuss each of the applicable areas of DSSA stages 4 and 5.

A piece of the architecture, referred to as the ASAC EA POC, was developed to prove the concept of the ASAC EA system. Figure 5-1 shows the context diagram of the entire system. The POC part of the system is shown by the highlighted box.

This section is built upon the *ASAC Executive Assistant Architecture Description Summary* and *Aviation System Analysis Capability Executive Assistant Design*, which covered DSSA stages 1, 2, 3, 4, and 5 (partial). The section will concentrate on the design, development, and acceptance of the POC.

The POC was successfully demonstrated to and accepted by NASA in March 1998.

Figure 5-1. POC Context Diagram



ASAC EA Requirements

Sixty-two requirements have been defined for the ASAC EA. Fifteen of the requirements applied to the ASAC EA POC and were tested as a part of ASAC EA POC development. These requirements are in bold. The ASAC EA POC requirements plus the requirements in normal text apply to the ASAC EA Beta version. Ten of the sixty-two requirements are in italics. They will be implemented for ASAC EA version 1.0.

The requirements listed in the following sections are grouped into nine areas. They are:

- Analysis Execution
- Analysis Management
- Analysis Specification
- Distributed Computing
- Error Handling
- General
- Model Specification
- Optimization
- Security.

Analysis Execution

- **AE0001** The Analyst shall have the capability to execute an analysis if an off-line administrator has granted the appropriate permissions.
- AE0002 The Analyst shall have the capability to view and modify model input data at user-defined intermediate steps in the analysis. Any modifications to the model inputs shall be logged.
  Note: In essence, provide the user with the capability to visually inspect and change data being transferred between models during the execution of an analysis.
- **AE0003** When an analysis is executed, the names of the models that are executed, as part of that analysis, will be logged to a log file.
- **AE0004** When an analysis is executed, its inputs and outputs will be logged.

- **AE0005** When a model is executed, its inputs and outputs will be logged.
- **AE0006** Upon completion of the execution of an analysis, the results will be presented to the user if the user is logged into the system.
- AE0007 Analysis and Model outputs shall be viewable in both raw and converted format.
- **AE0008** ASAC will provide a message to the user indicating a rough estimated time required to execute an analysis. Note: This will be a very rough estimate, as there are currently no plans to perform an interrogation of network and system(s) loading at the time of execution to provide a better estimate, not to mention the affect of data set size on model execution time.
- AE0009 ASAC EA shall support the execution of analyses in the "background" after users have logged off of the system.
- AE0010 The ASAC EA shall optionally mail a notification of analysis completion or suspension to the user, if the user is not logged into the system.
- AE0011 Users shall be able to cancel the execution of an analysis at any user pre-defined intermediate step.
- AE0012 Users shall be able to log back in and check the progress of, or cancel "active" analyses for which they have the appropriate permissions. When an analysis finishes, it shall remain "active" until the users views its outputs.
- AE0013 Analyses can be restarted from the beginning after their execution has finished or been canceled.
- AE0014 Users shall be able to set breakpoints on any data relationship. Breakpoints shall be settable before or after data conversion occurs in the data relationship.
- AE0015 Users shall be able to set preferences regarding e-mail delivery of various status messages that can get sent when they are not logged into the system.

Analysis Management

- **AM0001** The capability shall be provided to create an analysis by using off-line tools.
- AM0002 The Analyst shall have the capability to view an existing analysis if an off-line administrator has granted the appropriate permissions.
- **AM0003** The capability shall be provided to update an analysis by using off-line tools.

- AM0004 The Analyst shall have the capability to delete an analysis if an off-line administrator has granted the appropriate permissions.
- AM0005 The Analyst shall have the capability to copy an analysis if an off-line administrator has granted the appropriate permissions.
- AM0006 The capability shall be provided to store an analysis to the server for private or public use by using off-line tools.
- AM0007 The Analyst shall have the capability to store the results of an analysis to the server for private or public use if an off-line administrator has granted the appropriate permissions.

Analysis Specification

- **AS0001** An analysis may contain one or more models or analyses.
- AS0002 Analyses may have default input values.
- **AS0003** Default analysis input values may be overridden by the user.

Distributed Computing

- **DC0001** ASAC will accommodate operation of its models at remote sites.
- DC0002 ASAC EA shall provide the capability to allow analysts to run more than one analysis concurrently.
- **DC0003** ASAC EA shall support the concurrent execution of more than one instance of the same analysis on the same or different machines.
- **DC0004** ASAC EA shall support the concurrent execution of more than one instance of the same model on the same or different machines.
- **DC0005** The physical location of the models shall be transparent to the ASAC EA.
- DC0006 ASAC EA shall support a distributed application server model which allows multiple clients and servers to be located on different physical host machines.
- DC0007 ASAC EA shall allow users to run more than one analysis simultaneously.

Error Handling

- EH0001 The user shall be notified if the web server is not available (handled by the browser)
- EH0002 The user shall be notified if the analysis server is not available.
- **EH0003** The user shall be notified if a model server is not available.
- EH0004 The user shall be notified if the analysis server encounters a failure during analysis execution.
- EH0005 The user shall be notified if a model server encounters a failure during model execution.
- EH0006 The user shall be notified if an invalid data type or value for analysis/model input is specified.
- EH0007 The user shall be notified if the database is not available or if a database access error is encountered.

General

- GE0001 The user application will have an intuitive graphical user interface that adheres to the IBM CUA standards.

Model Specification

- MS0001 Models shall have valid default values upon initialization (when added to an analysis).
- MS0002 An off-line administrator shall have the capability to add new models to the system by:

  Developing (or adding developed) models that match a well-defined interface.

  Creating model specifications in a to be determined database that specifies the model parameters. e.g. inputs, outputs, and description.

  Writing and adding model wrappers that translate/map the well-defined model interface data element sets (DESs) to the model-specific interface for the model being added to the system (i.e., translators from DESs to model inputs and translators from model outputs to DESs).

- MS0003 Models may have default input values.
- MS0004 Default Model input values may be overridden by the user.
- MS0005 EA model inputs may be an ASCII file.

Optimization

Note: Optimization requirements will not be implemented for the beta, but will be designed and the necessary hooks will be implemented to support implementing it at some point in the future if required.

- *OP0001* The optimizer shall determine the number of times an analysis needs to run in order to achieve the specified goal.
- *OP0002* The EA shall provide an optimizing tool that allows users to specify a goal (e.g., minimize analysis output X) and let the system vary a given set of inputs to achieve the goal.
- *OP0003* When an analysis is rerun, the number of models that need to be rerun will be minimized based on the inputs that have changed. For example, given five models A, B, C, D, and E, where A = f(B + C) and B = f(D + E), when an input of E is changed, only B will be rerun (C and D will not be rerun).
- *OP0004* Optimization can only take place on an analysis, not an individual model or arbitrary set of models.
- *OP0005* The optimizer shall create a log containing convergence history and other relevant information which can be presented to the user for review.
- *OP0006* The ASAC EA shall support running an analysis a specified number of times and using a different set of pre-specified inputs for each iteration (i.e., table generation).

Security

- SE0001 An off-line system administrator will define the level of authorization for analyses and scenarios on a per-user or per-group basis.
- SE0002 The owning user shall have permissions to view and execute an analysis if an off-line administrator has granted the appropriate permissions.
- *SE0003* The owning user shall have the permission to grant or revoke view, execute, delete permissions to other users, provide that the owning user has "View," "Execute", and "Delete" permissions, respectively.
- *SE0004* The owning user shall be able to transfer ownership to other users.

- SE0005 An off-line administrator shall control user access to models.
- SE0006 Users must log into the system.
- SE0007 User authentication must be at least as secure as HTTP basic authentication.
- *SE0008* It shall be "difficult" for people to run ASAC EA models from outside the ASAC EA system, i.e., there must be no backdoors which allow unauthorized people to run models.
- *SE0009* Model servers shall be passed user authentication information so that they can control access (authorization) to specified users or groups.
- SE0010 An off-line administrator shall be able to define groups of users for authorization. Users can belong to multiple groups.
- SE0011 Scenarios will have "Read," "Write," and "Delete" permissions associated with them. Analyses will only have "Read "permissions. Anybody able to read an analysis can create a scenario for that analysis and execute it.

POC Goals

The ASAC EA POC was developed to address the high risk areas of the ASAC EA system. The goals of the ASAC EA POC were to

- Demonstrate seamless integration of standalone models
- Output of models automatically feeds input of next model(s) in analysis
- Demonstrate integration of models on multiple machines
- Validate system design.

Review and Iterate DSSA Substage 2-8: Define Assumptions

The assumptions defined in the design phase of the project and described in the *Aviation System Analysis Capability Executive Assistant Design* still apply. In addition, the following assumptions were made during the implementation of the POC:

- An Analysis owns the models and relationships that it contains. When an analysis is deleted, the models and relationships it contains are deleted as well.

- All the changeable inputs to a model can be represented as a set of named data elements. Each data element implements a heterogeneous table, e.g., each column can be a different data type. This allows each data element to represent a scalar value, an array, a record, or a complex table. Each input to a model must fit into this form.

- Analyses will be created off-line, not through a GUI. Users will not be able to create their own analyses.

- Analyses will specify default (fixed) values for all model variables and will specify which values can be changed by the users, e.g, analysis variables. Although users can view the values of model variables, they cannot modify them. Only analysis variables can be changed. This eliminates the need for a custom user interface for each model.

- When checkpoints are implemented, only variables predetermined by the analysis specification can have their values modified between analyses.

- Users cannot add or delete the models in an analysis. Users cannot add or delete data relationships in an analysis.

REVIEW AND ITERATE DSSA SUBSTAGE 2-9: DEFINE ISSUES

Issues remaining from the *Aviation System Analysis Capability Executive Assistant Design* are as listed below. Answers have been provided where an issue has been resolved:

- How does the EA system handle or detect non-termination of models?

  There are two conditions that could cause non-termination of a model and, therefore, analysis deadlock. The first is the model completes but does not return all necessary data for the model to be considered complete. This is easily detected in the code. The second condition is the model never returns but instead gets caught in an infinite loop caused by poor model design or implementation. This case is more difficult to detect, and methods will be investigated during the next phase of development. All other non-termination conditions should be detected and handled as errors.

- How is data passed among components? Pass the data or data file name?

  Data is passed directly between components to avoid having to depend on multiple communication protocols, i.e., CORBA plus a file transfer protocol.

- Should multiple processes be spawned for the analysis application, or should there be separate invocations of the program?

For the ASAC EA POC, separate invocations of the Analysis application exist. For the Beta version and full ASAC EA system, each analysis will be a separate process (or thread) spawned by the analysis server application.

- What are the space constraints on user systems (maximum size for the user application)?

  Space constraints will be examined during next phase of development.

- What is the target size of the analysis application?

  The target size will be examined during next phase of development.

New issues have been identified during ASAC EA POC development. They are:

- Should customer proprietary models be hosted on LMI servers?

  Customer proprietary models will not be hosted on LMI servers in order to minimize the cost of implementing ASAC security and to maximize customer protection. Customers will maintain and host proprietary models in their respective organizations so the models will be subject to their internal security protection. Interaction between ASAC and proprietary models hosted on customer servers not collocated at LMI will be enabled via file/database transfer from the customer to ASAC and return, and/or appropriate customer legacy model wrapper and ORB software. Customer models hosted on LMI servers will be protected to the same level of security as all other models hosted on the LMI servers.

- In the absence of a currently available security solution, such as a CORBA level 2 security service, is providing a completely secure environment in a distributed system such as ASAC EA infeasible. What security should be provided for the ASAC EA system?

  The level of security for the ASAC EA system will be username and password passed over the network. This is the level of protection offered by the majority of internet-based applications., i.e., telnet, ftp, http, and pop3.

- Is optimization at the analysis level (across models) necessary?

  We will not implement multimodel optimization in the ASAC EA POC or Beta version; however, we will not preclude optimization from being implemented in the future (we will consider optimization in our high-level design so future implementation will not be disruptive to the system).

- Do we use a database or some other mechanism (flat files) for storing analysis and model specifications? If we use a database, is it relational, OO, or a hybrid?

Storage mechanisms will be examined during next phase of development.

- Given the name of the data transformer specification, how does the program determine if it is a model specification or analysis specification without detailed understanding of the differences between those two files?

  The specification identification will be examined during next phase of development.

- When a model fails because of an error, how is the model's parent analysis notified? (How do we handle errors in a multithreaded environment?)

  Error notification will be examined during next phase of development.

DSSA Stage 4—Refine POC Domain Models

Domain models that were developed in DSSA substages 4-3 to 4-8 of the domain-engineering process and documented in the *Aviation System Analysis Capability Executive Assistant Design* were refined during POC development. The domain models that were refined are:

- 4-3 Use case diagrams
- 4-4 Interaction diagrams
  - Sequence diagrams
  - Collaboration diagrams
- 4-5 Package diagrams
- 4-6 Class diagrams
- 4-7 State diagrams
- 4-8 Deployment diagrams

Thirteen classes were defined and described during POC design. They are:

- Subject,
- Observer,
- DataTransformer,
- Analysis,
- AnalysisSpec,
- Model,
- ModelSpec,
- DataRelationship,
- DataRelationshipSpecification,

- DataElement,
- DataElementSet,
- DataConverter, and
- DataStorage.

A number of new classes were added during implementation of the POC. The majority of the classes are infrastructure or utility classes, which help encapsulate required functionality. Some classes were added to encapsulate and factor out common functionality that existed in more than one previously defined class.

The new classes are:

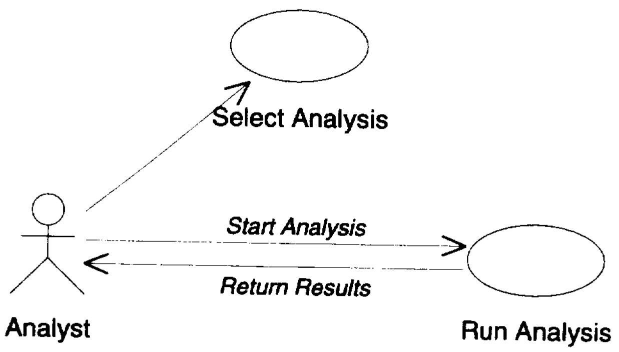
- Log,
- Application,
- CorbaClient,
- CorbaServer,
- DataStorage,
- Specification,
- Scanner,
- TransformerSpec,
- ModelSpec,
- AnalysisSpec,
- Thread,
- Mutex,
- AnalysisClient,
- ModelWrapper,
- ModelWrapper_i,
- ModelServer,
- DataElementIterator,
- Evaluate.

These classes will be discussed in more detail throughout this chapter.

DSSA Substage 4-3: Develop Use Case Diagrams

Use Case diagrams are used to show a typical interaction between a user and the system. The Use Case diagram for the POC is shown in Figure 5-2. It illustrates that a user will be able to select an analysis, start an analysis, and obtain the results from the analysis.

Figure 5-2. POC Use Case Diagram



DSSA Substage 4-4: Develop Interaction Diagrams

Interaction diagrams are diagrams that describe how groups of objects collaborate. These diagrams usually capture the behavior of a single Use Case. The two types of Interaction diagrams are sequential diagrams and collaboration diagrams. Sequential diagrams and Collaboration diagrams give the same temporal information, but are shown in two different ways. Objects in a sequence diagram are shown as a box with a dashed line below it that represents the objects lifeline.

Each message is represented by an arrow between two lifelines. Objects in a collaboration diagram are shown as icons and the message is represented by arrows between two icons.

Interaction diagrams were developed for four areas. They are:

- Building an Analysis,
- Building a Model,
- Building a DataRelationship between two DataTransformers,
- Running an Analysis.

BUILDING AN ANALYSIS

Figure 5-3. Building an Analysis Sequence Diagram

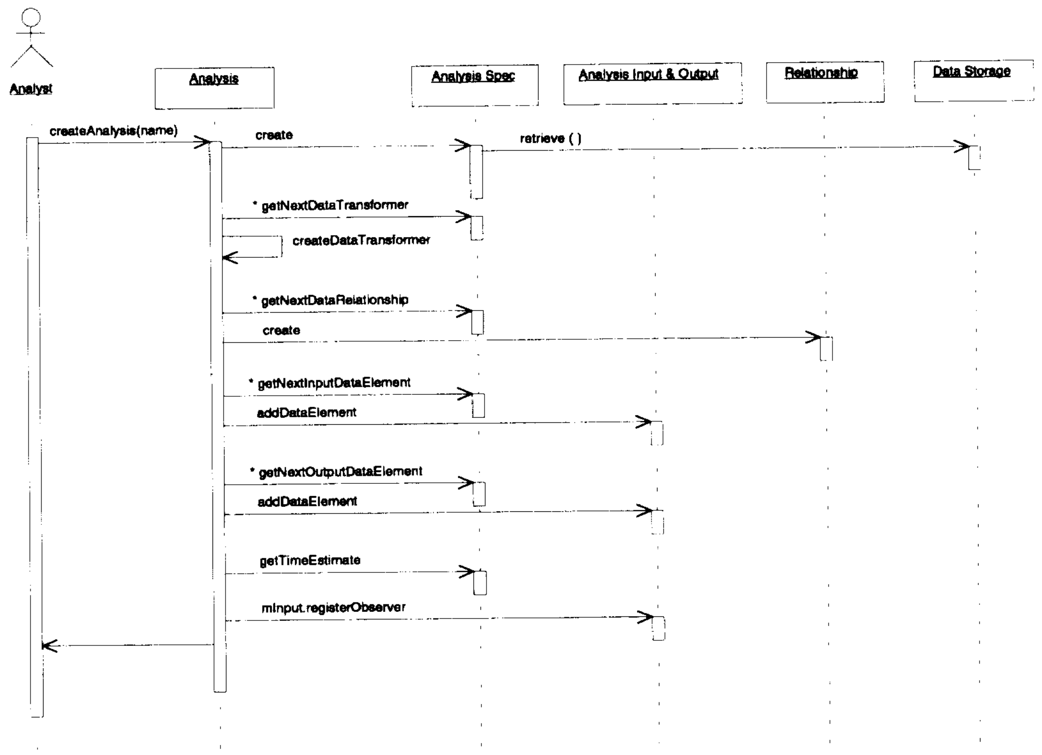
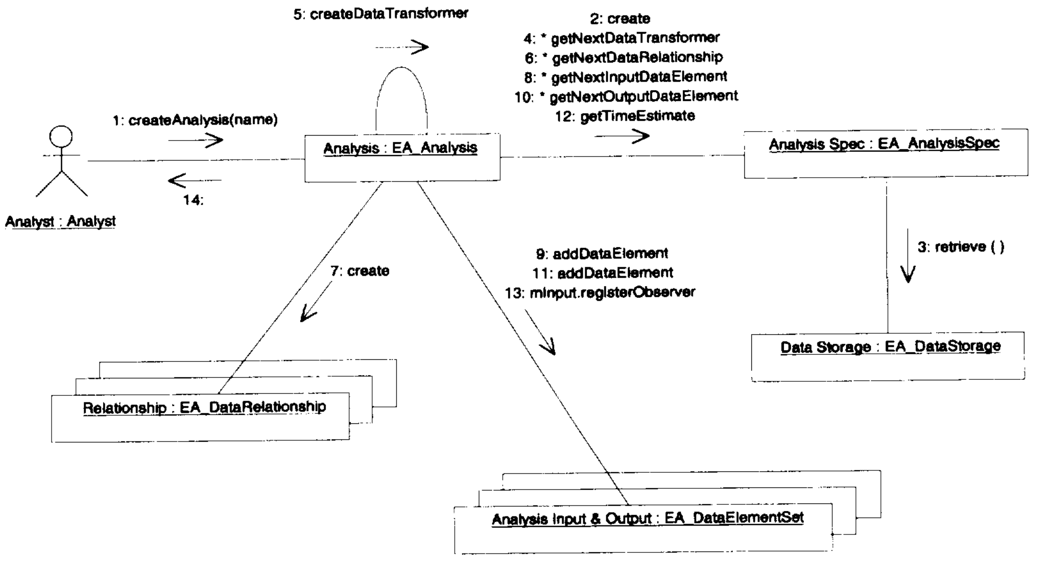


Figure 5-4. Building an Analysis Collaboration Diagram


5: createDataTransformer
2: create
4: * getNextDataTransformer
6: * getNextDataRelationship
8: * getNextInputDataElement
10: * getNextOutputDataElement
12: getTimeEstimate
1: createAnalysis(name)
14:
Analyst : Analyst
Analysis : EA_Analysis
Analysis Spec : EA_AnalysisSpec
3: retrieve ( )
7: create
9: addDataElement
11: addDataElement
13: mInput.registerObserver
Data Storage : EA_DataStorage
Relationship : EA_DataRelationship
Analysis Input & Output : EA_DataElementSet

BUILDING A MODEL

Figure 5-5. Building a Model Sequence Diagram

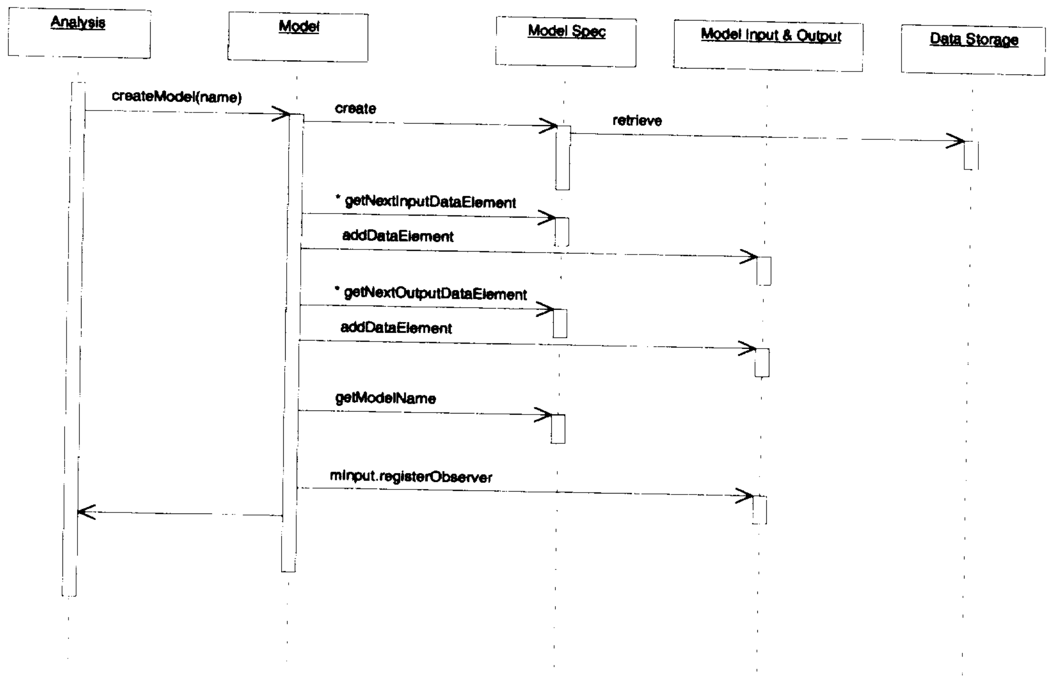
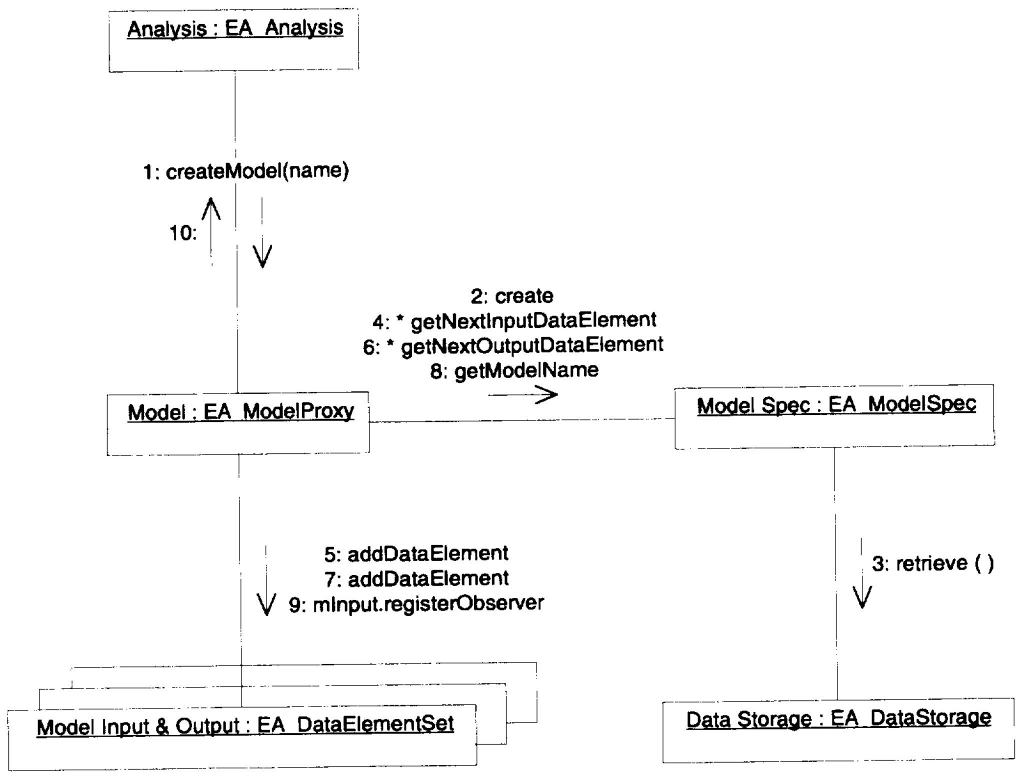


Figure 5-6. Building a Model Collaboration Diagram


Analysis : EA Analysis
1: createModel(name)
10:
2: create
4: * getNextInputDataElement
6: * getNextOutputDataElement
8: getModelName
Model : EA ModelProxy
Model Spec : EA ModelSpec
5: addDataElement
7: addDataElement
9: mInput.registerObserver
3: retrieve ( )
Model Input & Output : EA DataElementSet
Data Storage : EA DataStorage

BUILDING A DATARELATIONSHIP BETWEEN TWO DATATRANSFORMERS

Figure 5-7. Building a DataRelationship Between Two DataTransformers Sequence Diagram

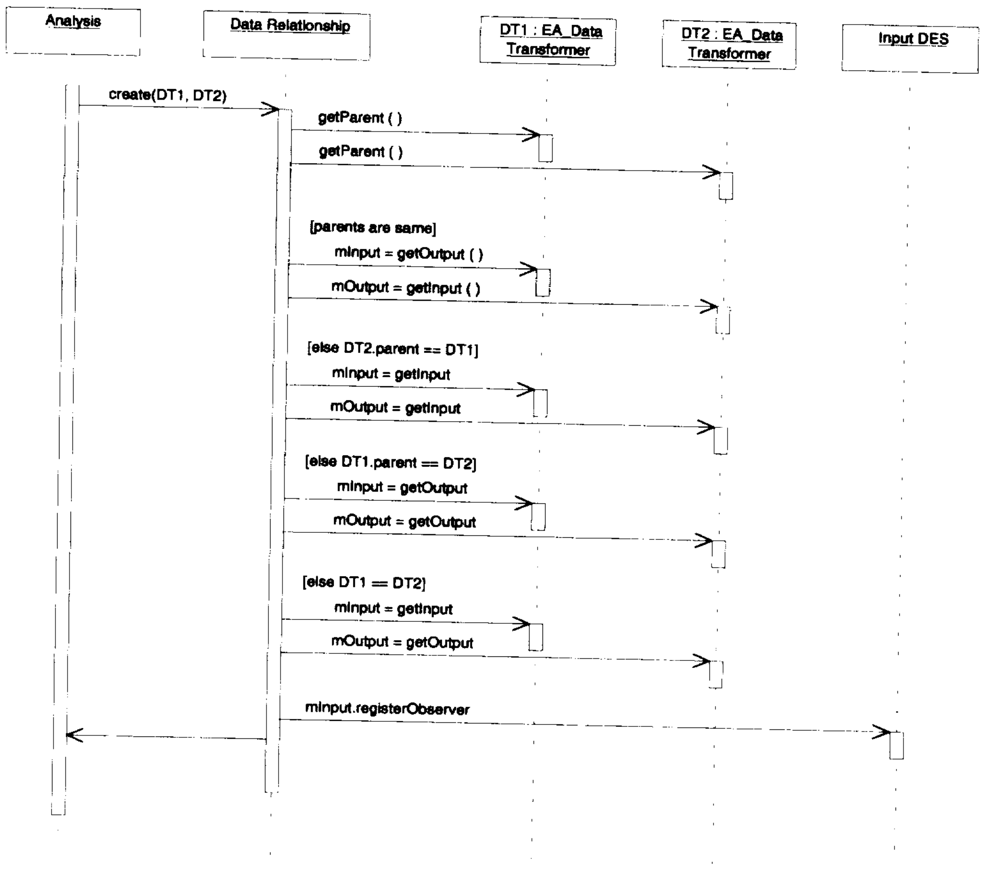
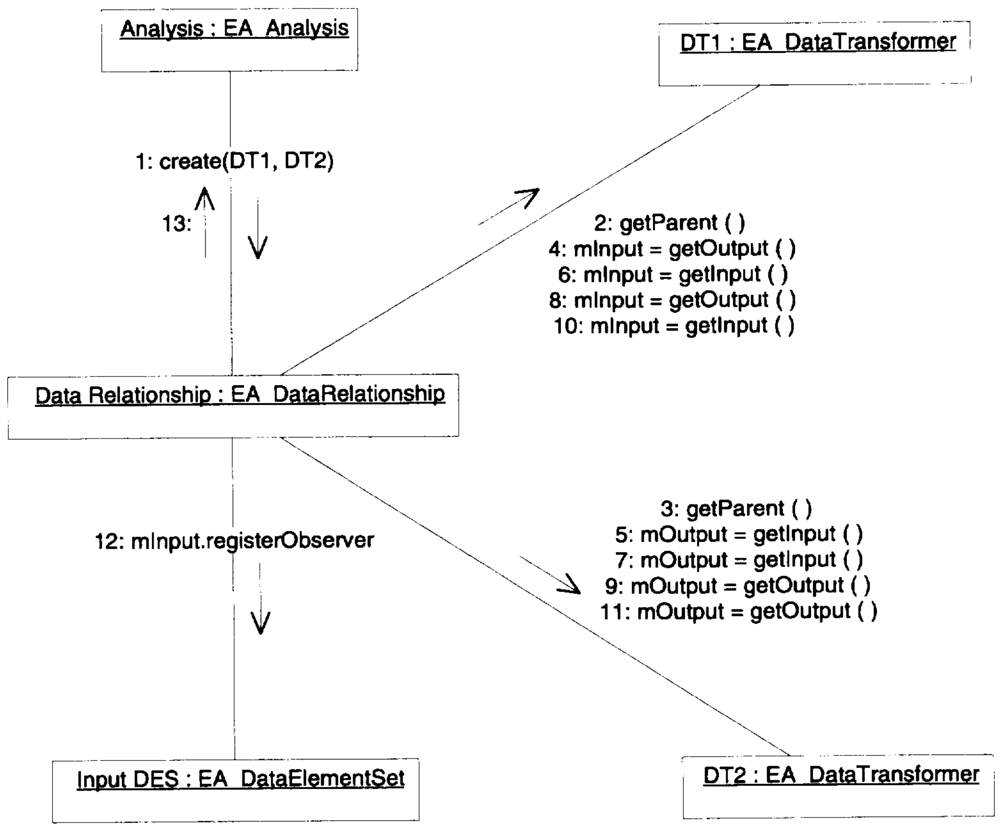


Figure 5-8. Building a DataRelationship Between Two DataTransformers Collaboration Diagram


Analysis : EA Analysis
DT1 : EA DataTransformer
1: create(DT1, DT2)
13:
2: getParent ( )
4: mInput = getOutput ( )
6: mInput = getInput ( )
8: mInput = getOutput ( )
10: mInput = getInput ( )
Data Relationship : EA DataRelationship
12: mInput.registerObserver
3: getParent ( )
5: mOutput = getInput ( )
7: mOutput = getInput ( )
9: mOutput = getOutput ( )
11: mOutput = getOutput ( )
Input DES : EA DataElementSet
DT2 : EA DataTransformer

RUNNING THE ANALYSIS

Figure 5-9. Running the Analysis Sequence Diagram

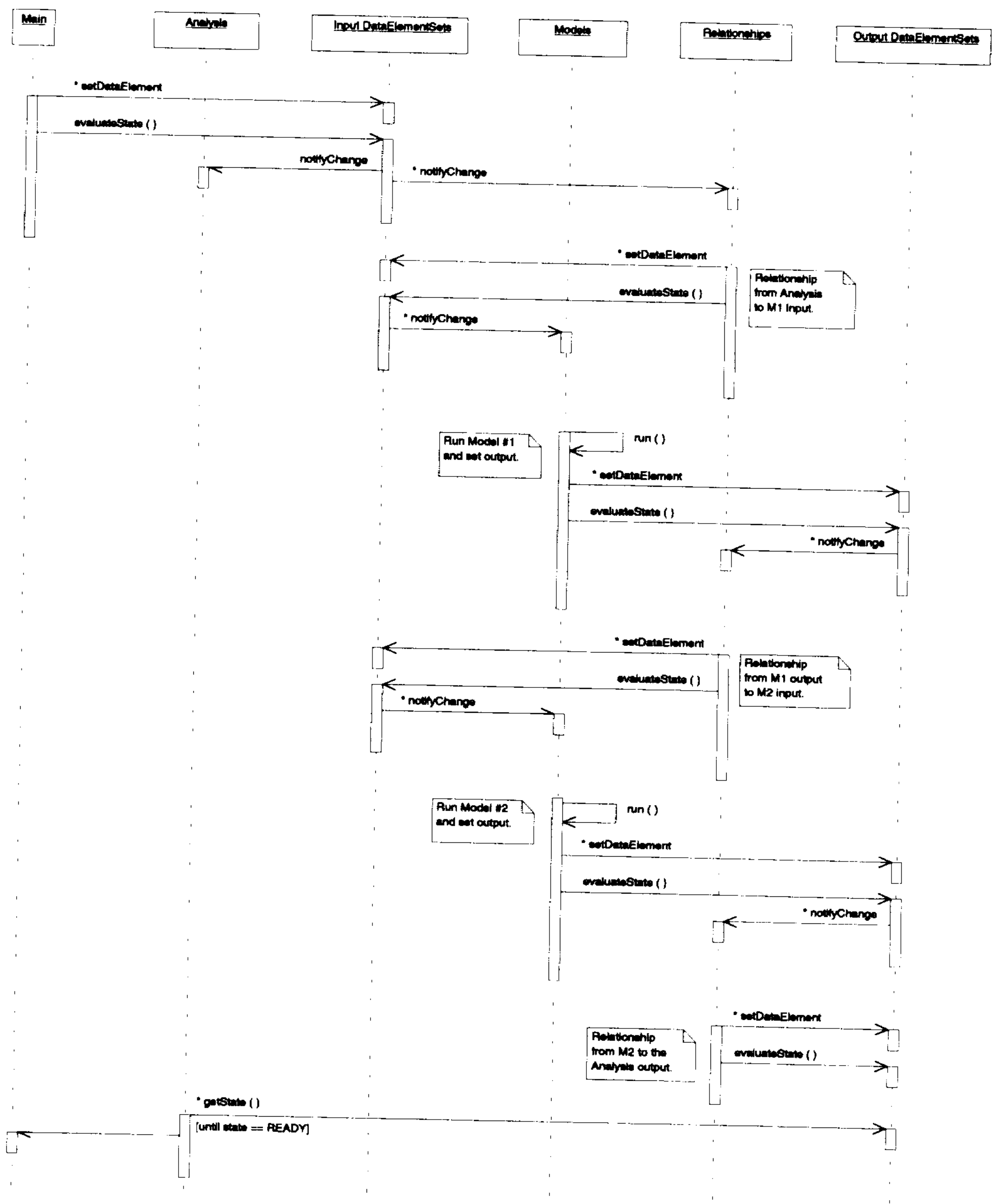
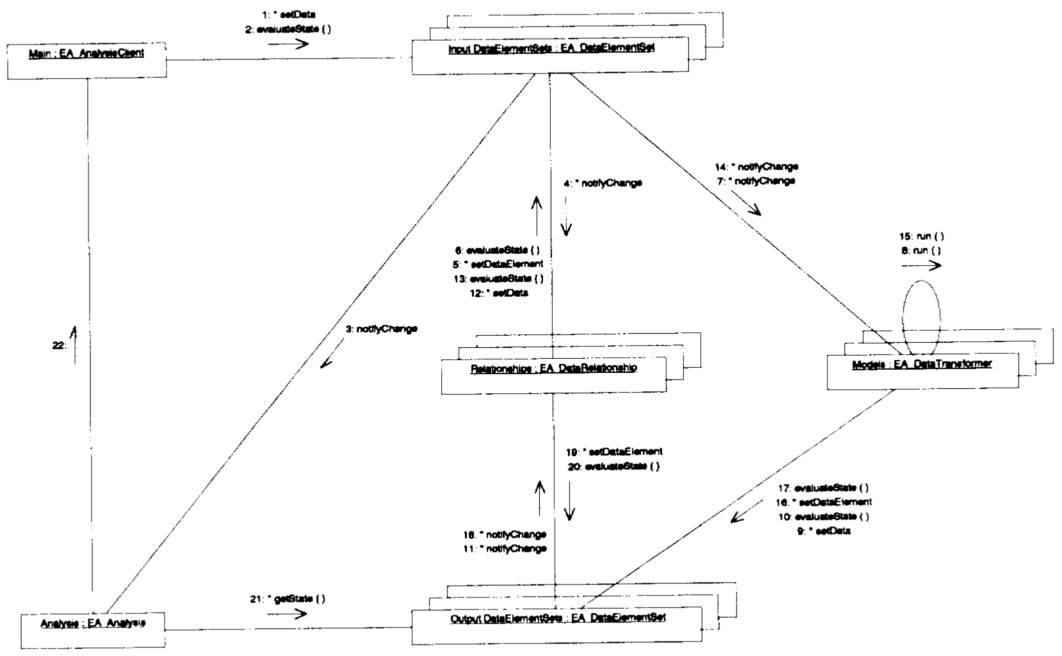


Figure 5-10. Running The Analysis Collaboration Diagram



DSSA Substage 4-5: Develop Package Diagrams

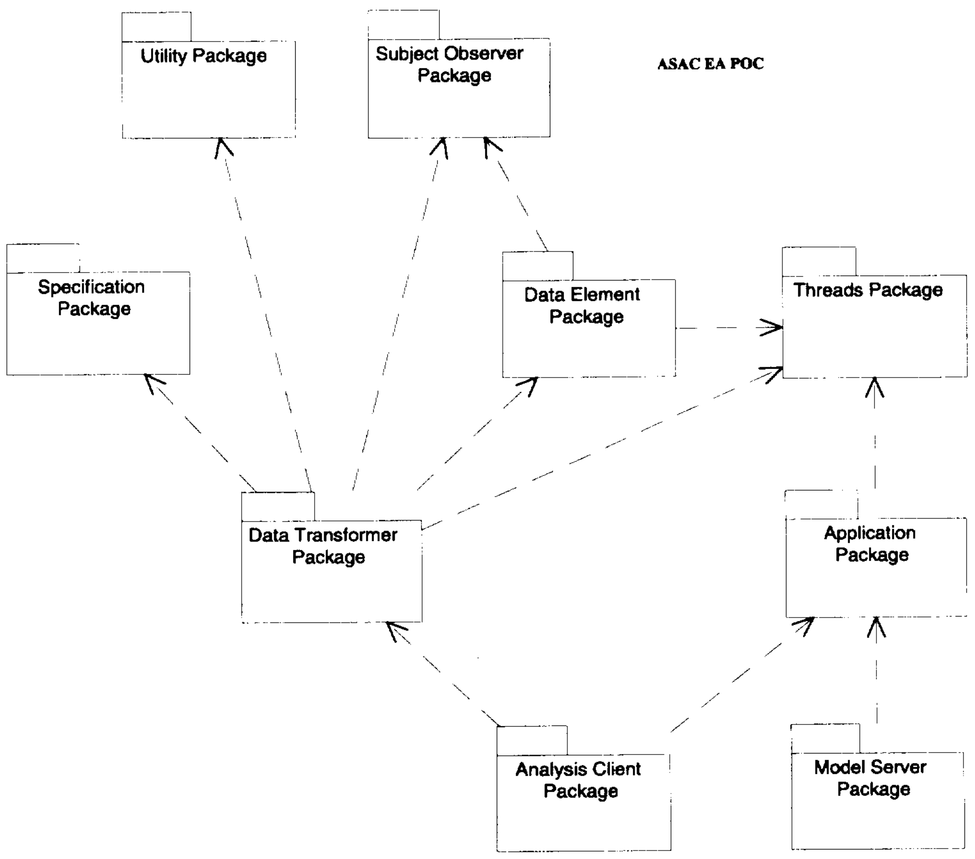
Package diagrams are used for readability purposes only. When a design becomes large, it is convenient to separate groups of classes into separate packages. The POC design has been divided into nine class packages:

- Subject Observer
- Specification
- Data Transformer
- Data Element
- Threads
- Utility
- Application
- Analysis Client
- Model Server.

Figure 5-11 shows the POC package diagram. The dependencies among the classes are denoted by the dashed lines. The dependencies are the following:

- The Data Transformer package depends on the Specification package to read in Analysis and Model specifications.
- The Data Transformer package depends on the Data Element package to hold the inputs and outputs for data transformers.
- The Data Transformer and Data Element packages depend on the Subject Observer package to notify Data Transformers, Data Relationships, and Data Element Sets of state changes in other objects that they depend on.
- The Data Transformer and Data Element packages depend on the Threads package to execute models in parallel and to provide synchronization between threads and mutually exclusive access to shared data.
- The Data Transformer package depends on the Utility package for miscellaneous utility functions.
- The Application package depends on the Threads package for implementing a thread-safe asynchronous signal-handling thread.
- The Analysis Client and Model Server packages depend on the Application package to handle basic application functions, such as signal handles, error login, command-line parsing, as well as functions specific to CORBA clients and servers, such as initialization and object registration.
- The Analysis Client package depends on the Data Transformer package to coordinate the execution of analyses consisting of multiple potentially distributed models.

Figure 5-11. Package Diagram



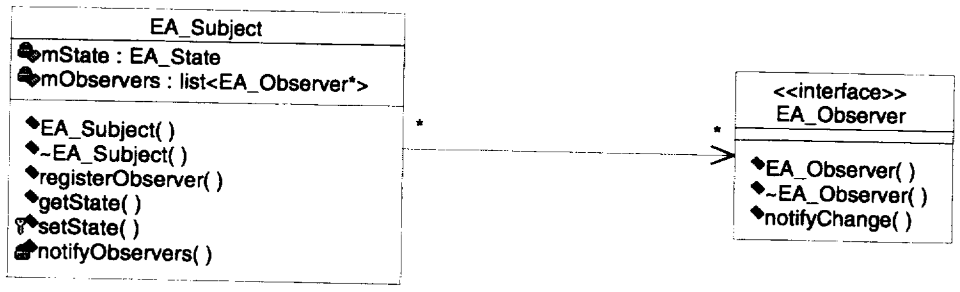
DSSA Substage 4-6: Develop Class Diagrams

Class diagrams are used to illustrate class models and their relationships with other classes. The class diagrams will be shown with their package.

SUBJECT OBSERVER PACKAGE

The Subject Observer package contains two classes, which define a subject-observer pattern (also called publish-subscribe or observer-observable). Each subject may have any number (zero or more) of observers and each observer may receive notifications from any number of subjects. The class diagram is shown in Figure 5-12.

Figure 5-12. Subject Observer Class Diagram



Subject

The subject is a superclass that defines the properties of an object being observed. A subject may have any number of dependent observers. All observers are notified when the subject undergoes a change in state. A list of properties and methods for this class can be found in Table 5-1.

Table 5-1. Properties and Methods for Subject Class

Private Properties	
mState : EA_State	The current state of the subject.
mObservers : list<EA_Observer*>	The list of observers that the subject notifies when its state changes.
Public Methods	
EA_Subject (initState : enum EA_State = WAITING) : EA_Subject	Constructor.
~EA_Subject () :	Destructor.
registerObserver (obs : EA_Observer&) : void	Registers an observer object as an observer of this subject.
getState () : enum EA_State	Gets the current state of the subject.
Protected Methods	
setState (newState : enum EA_State) : void	Sets the state of the subject, and notifies its observers that the state has changed by calling the notifyChange method on each observer.

Observer

The Observer defines an updating interface for objects that should be notified of changes in a subject's state. In response to notification, observers query the subject to synchronize its state with the subject's state. A list of properties and methods for this class can be found in Table 5-2.

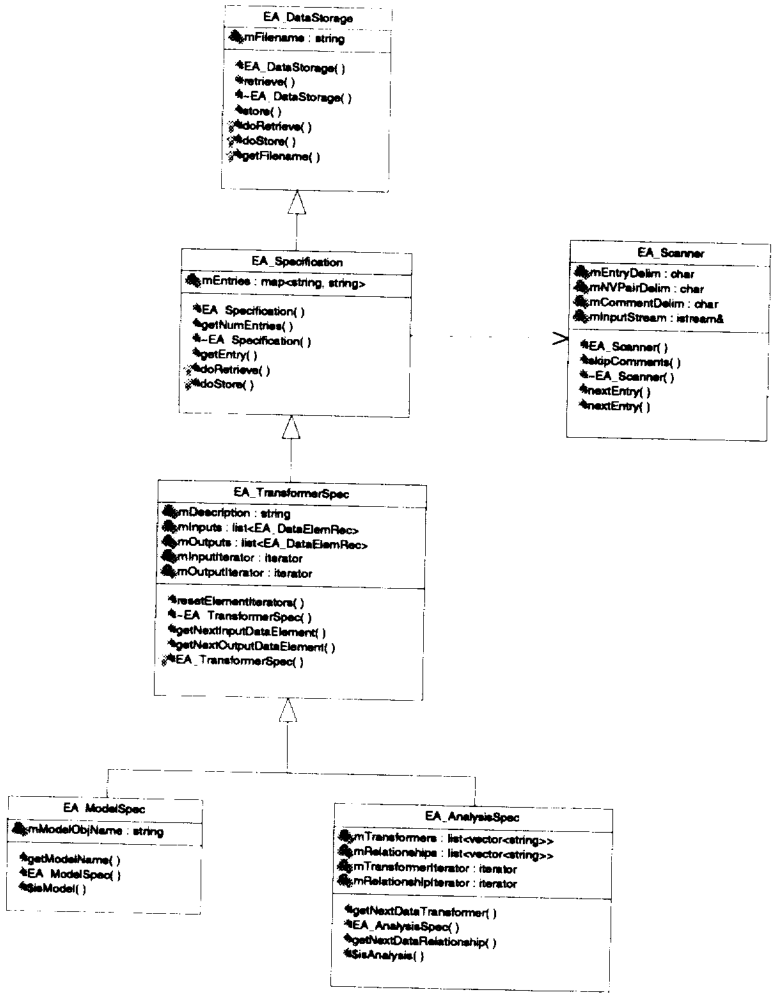
Table 5-2. Properties and Methods for Observer Class

Public Methods	
EA_Observ er () : EA_Observer	Constructor.
~EA_Observer () :	Destructor.
notifyChange (subj : EA_Subject&) : void	Called by a subject to notify its observers that its state has changed.

SPECIFICATION PACKAGE

The Specification package contains classes that support reading and writing of analysis and model specifications. An analysis specification is a file or other unit of data that contains the information (e.g., inputs, outputs, names of models, relationships) necessary for creating an analysis that can be executed. Similarly, a model specification contains the information necessary for creating and executing a model as part of an analysis. The class diagram is shown in Figure 5-13.

Figure 5-13. Specification Package Class Diagram



DataStorage Class

The DataStorage class provides an interface to a data stream and enables objects to store or retrieve themselves from the stream. The stream could be stored in a file, an entry in a database, etc. Objects that inherit from the DataStorage class must define exactly how the object is stored and retrieved. A list of properties and methods for this class can be found in Table 5-3.

Table 5-3. Properties and Methods for DataStorage Class

Private Properties	
mFilename : string	The name of the file to read data from and write data to.
Public Methods	
EA_DataStorage (filename : const RWCString&) : EA_DataStorage	Constructor.
retrieve () : void	Operation that retrieves data objects. Calls doRetrieve on the sub-class and passes it the input stream.
~EA_DataStorage () :	Destructor.
store () : void	Operation that storess data objects. Calls doStore on the subclass and passes it the output stream.
Protected Methods	
doRetrieve (input : istream&) : void	Virtual function that defines how to read the data from the input stream. Must be defined by the subclass.
doStore (output : ostream&) : void	Virtual function that defines how to write the data to the output stream. Must be defined by the subclass.
getFilename () : const RWCString&	Returns the filename that the object is stored to and retrieved from.

Specification Class

The Specification class defines the basic format for all specifications and defines how the specifications are stored and retrieved via the DataStorage interface that it inherits. A specification is made up of one or more sections or entries that it parses using the Scanner class. The “getEntry” method provides access to the data in each section of the specification. A list of properties and methods for this class can be found in Table 5-4.

Table 5-4. Properties and Methods for Specification Class

Private Properties	
mEntries : map<string, string>	The data associated with each section (entry) of the specification, stored as name/value pairs.
Public Methods	
EA_Specification (name : const RWCString&) : EA_Specification	Constructor.
getNumEntries () : int	Returns the number of entrie (named sections) that
~EA_Specification () :	Destructor.
getEntry (name : const RWCString&, value : RWCString&) : bool	Returns the data associated with the specified section (entry) of the specification.
Protected Methods	
doRetrieve (input : istream&) : void	Reads and parses the specification from the given input stream.
doStore (output : ostream&) : void	Writes the specification to the given output stream.

Scanner Class

The Scanner class provides the ability to parse an input stream that contains name/value pairs in a specified format. The Scanner class is used by the Specification class to parse model and analysis specifications, as well as by the DataRelationship class to process data relationship specification files. A list of properties and methods for this class can be found in Table 5-5.

Table 5-5. Properties and Methods for Scanner Class

Private Properties	
mEntryDelim : char	The delimiter that specifies the end of a section or entry.
mNVPairDelim : char	The delimiter between the name of a section or entry and its value.
mCommentDelim : char	The delimiter which indicates the beginning of a comment.
mInputStream : istream&	The input stream which the scanner reads from.
Public Methods	
EA_Scanner (input : istream&, entry : char = ';', nvPair : char = '=', comment : char = '#') : EA_Scanner	Constructor.
skipComments () : istream&	Skips blank lines and lines that begin with the comment delimiter.
~EA_Scanner () :	Destructor.
nextEntry (name : RWCString&, value : RWCString&) : istream&	Gets the name and value of the next entry in the specification and returns true if a complete entry was found.
nextEntry (value : RWCString&) : istream&	Reads data (skipping comments) until an end of entry delimiter is found.

TransformerSpec Class

The TransformerSpec class inherits from the Specification class and encapsulates the understanding of entries that are common to both model and analysis specifications. In particular, the TransformerSpec class parses the sections that define the input and output data elements for a model or an analysis. A list of properties and methods for this class can be found in Table 5-6.

Table 5-6. Properties and Methods for TransformerSpec Class

Private Properties	
mDescription : string	A description of the data transformer.
mInputs : list<EA_DataElemRec>	A list of the inputs that this data transformer requires.
mOutputs : list<EA_DataElemRec>	A list of the outputs that this data transformer produces.
mInputIterator : iterator	Iterator used to iterate over the list of input data elements.
mOutputIterator : iterator	Iterator used to iterate over the list of output data elements.
Public Methods	
resetElementIterators () : void	Resets the iterators to the beginning of the lists.
~EA_TransformerSpec () :	Destructor.
getNextInputDataElement (element : EA_DataElemRec_t&) : bool	Gets the next input data element from the specification.
getNextOutputDataEle-ment (element : EA_DataElemRec_t&) : bool	Gets the next output data element from the specification.
Protected Methods	
EA_TransformerSpec (name : const RWCString&) : EA_TransformerSpec	Constructor.

ModelSpec Class

The ModelSpec class manages specification data for a particular model. A list of properties and methods for this class can be found in Table 5-7.

Table 5-7. Properties and Methods for ModelSpec Class

Private Properties	
mModelObjName : string	The name of the CORBA model object to call in order to run this model.
Public Methods	
getModelName () : const RWCString&	Returns the name of the CORBA model object to call to run this model.
EA_ModelSpec (name : const RWCString&) : EA_ModelSpec	Constructor.
isModel (name : const RWCString&) : bool	Returns true if the specified string is the name of a model specification.

AnalysisSpec Class

The AnalysisSpec class manages specification data for a particular analysis. A list of properties and methods for this class can be found in Table 5-8.

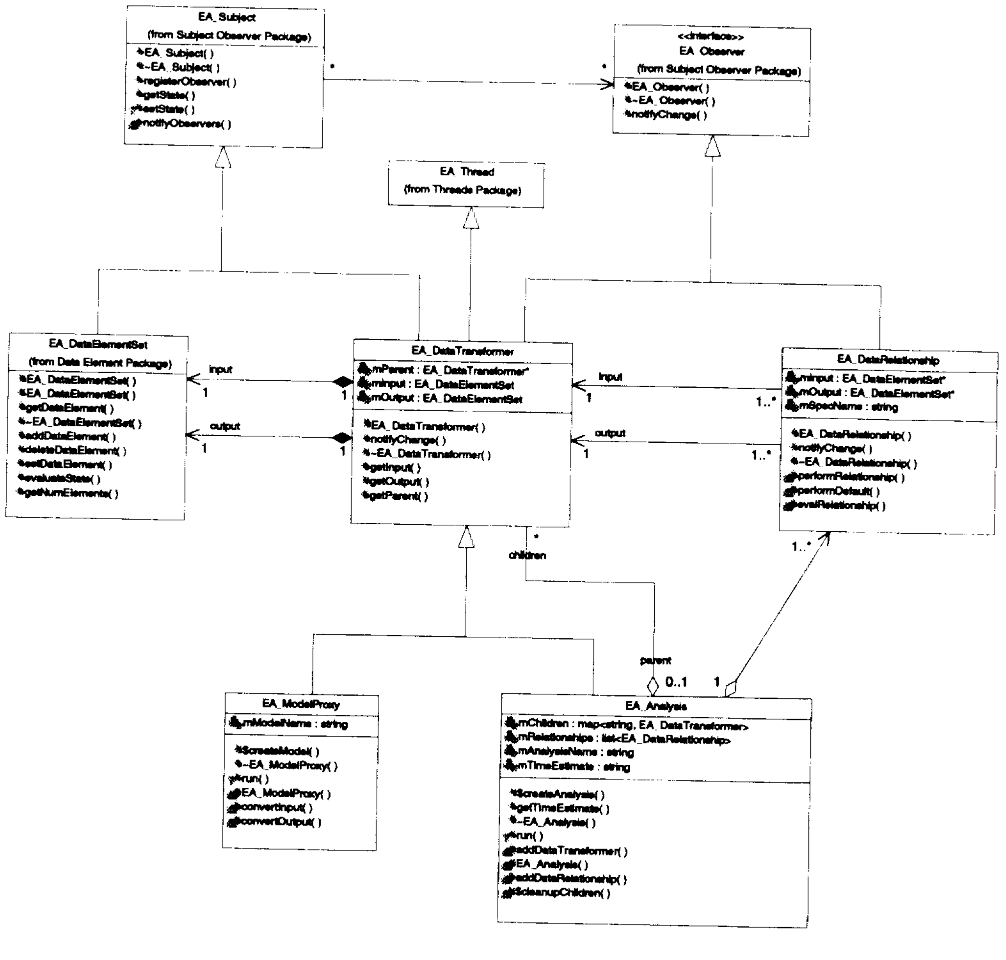
Table 5-8. Properties and Methods for AnalysisSpec Class

Private Properties	
mTransformers : list<vector<string>>	The list of data transformers that make up the analysis.
mRelationships : list<vector<string>>	The list of data relationships that the analysis contains.
mTransformerIterator : iterator	Iterator used to iterate over the list of data transformers.
mRelationshipIterator : iterator	Iterator used to iterate over the list of data relationships.
Public Methods	
getNextDataTransformer (name : RWCString&, id : RWCString&) : bool	Gets the name of the next data transformer that is part of the analysis from the specification.
EA_AnalysisSpec (name : const RWCString&) : EA_AnalysisSpec	Constructor.
getNextDataRelationship (from : RWCString&, to : RWCString&, name : RWCString&) : bool	Gets the next data relationship that is part of the analysis from the specification.
isAnalysis (name : const RWCString&) : bool	Returns true if the specified string is the name of an analysis specification.

DATA TRANSFORMER PACKAGE

The Data Transformer package contains the classes that execute an analysis. The Subject, Observer, DataElementSet, and Thread classes are shown in the class diagram to illustrate their relationships with the classes in this package. They are not a part of the Data Transformer package. All DataTransformers inherit from the Subject and Observer classes and contain two DataElementSets; one acts as its input, and the other its output. An Analysis is a special kind of DataTransformer, which contains other DataTransformers (models and analyses) and DataRelationships. DataRelationships act as links in an analysis that pass data from one DataTransformer to the next. The class diagram for the Data Transformer package is shown in Figure 5-14.

Figure 5-14. Transformer Class Diagram



DataTransformer

The DataTransformer is an abstraction for a class that transforms input data values into output data values. The class has two DataElementSets that contain the input and output values of the transformer. The DataTransformer "watches" its input DES and automatically performs the transformation when all of its inputs have been set (i.e., the input DES changes to the "ready" state). A list of properties and methods for this class can be found in Table 5-9. The DataTransformer inherits from the Subject and Observer classes, so it also contains the properties and methods shown in Tables 5-1 and 5-2.

Table 5-9. Properties and Methods for DataTransformer Class

Private Properties	
mParent : EA_DataTransformer*	A pointer to the parent of this data transformer or NULL if it has no parent.
mInput : EA_DataElementSet	The set of data elements that acts as the input to the data transformer.
mOutput : EA_DataElementSet	The set of data elements that acts as the output of the data transformer.
Public Methods	
EA_DataTransformer (spec : EA_TransformerSpec&, parent : EA_DataTransformer*) : EA_DataTransformer	Constructor. Initializes the input and output DESes based on the TransformerSpec. Also registers as an observer to its input DES.
notifyChange (subj : EA_Subject&) : void	Virtual method from EA_Observer. If the input DES is ready, start a thread and run the data transformer in it.
~EA_DataTransformer () :	Destructor.
getInput () : EA_DataElementSet&	Returns a reference to the data transformer's input DES.
getOutput () : EA_DataElementSet&	Returns a reference to the data transformer's output DES.
getParent () : EA_DataTransformer*	Returns a pointer to the transformer's parent analysis, or null if there isn't one.

Analysis

The Analysis class is a data transformer that is made up of other data tranformers (models or other analyses). The class manages the creation and instantiation of its data transformers and the data relationships between them. A list of properties and methods for this class can be found in Table 5-17. The Analysis class inherits from the DataTransformer class, so it also contains the properties and methods shown in Tables 5-1, 5-2, and 5-9.

Table 5-10. Properties and Methods for Analysis Class

Private Properties	
mChildren : map<string, EA_DataTransformer>	A list of data transformers that make up the body of the analysis. Each transformer has a name associated with it.
mRelationships : list<EA_DataRelationship>	The list of relationships that are part of this analysis.
mAnalysisName : string	The name of the analysis.
mTimeEstimate : string	A rough estimate of the time that the analysis will take to execute.
Public Methods	
createAnalysis (name : const RWCString&, parent : EA_DataTransformer* = NULL) : EA_Analysis*	Factory method used to create an analysis given an analysis specification name.
getTimeEstimate () : const RWCString&	Returns an estimate of the time that this analysis is expected to take.
~EA_Analysis () :	Destructor.
Protected Methods	
run () : void	Runs the analysis. Once the analysis has started, waits until its output DES becomes ready (i.e. all models have finished and written their output).

Model

The Model class is a DataTransformer that acts as an interface or proxy to a distributed model application. When the time comes for the Model class to transform its input data, the Model class passes its inputs via CORBA to a model that runs on a separate machine. A list of properties and methods for this class can be found in Table 5-11. The Model class inherits from the DataTransformer class, so it also contains the properties and methods shown in Tables 5-1, 5-1, and 5-9.

Table 5-11. Properties and Methods for Model Class

Private Properties	
mModelName : string	The name of the CORBA model object to call in order to run this model.
Public Methods	
createModel (name : const RWCString&, parent : EA_DataTransformer*) : EA_ModelProxy*	Factory method used to create a Model given a Model Specification name.
~EA_ModelProxy () :	Destructor.
Protected Methods	
run () : void	Runs the model by passing the inputs from the input DES to a CORBA model object and storing its output in the output DES.

DataRelationship

The DataRelationship class acts as a link between two DataTransformers. It waits for the one of the DES of the input DataTransformer to be set, gets its data values, performs any necessary data transformation or conversion, and sets the values in the one of the DES of the target DataTransformer. A list of properties and methods for this class can be found in Table 5-12.

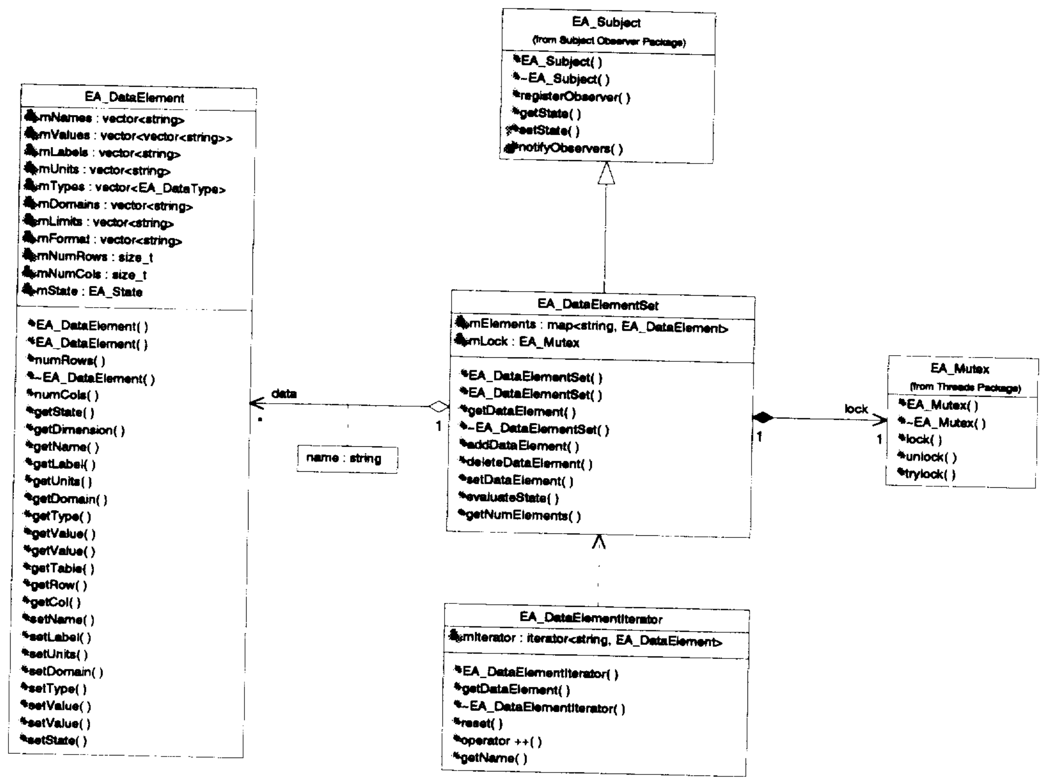
Table 5-12. Properties and Methods for DataRelationship Class

Private Properties	
mInput : EA_DataElementSet*	The DES which acts as the input to the data relationship.
mOutput : EA_DataElementSet*	The DES that the data relationship writes its output to.
mSpecName : string	The name of the data relationship specification (if any) to use for this relationship.
Public Methods	
EA_DataRelationship (in : EA_DataTransformer&, out : EA_DataTransformer&, specName : const RWCString& = "") : EA_DataRelationship	Constructor.
notifyChange (subj : EA_Subject&) : void	If the input DES is ready, perform the relationship and set the values of the output DES.
~EA_DataRelationship () :	Destructor.

DATA ELEMENT PACKAGE

The Data Element package contains the DataElementSet, DataElement, and DataElementIterator classes and their relationships. The Subject and Mutex classes do not belong to this package, but are shown in this package to illustrate their relationship to the DataElementSet class. The class diagram illustrates that a DataElementSet contains zero or more DataElements and that each DataElement has a string associated with it which represents its name within the DataElementSet. The class diagram for the Data Element package is shown in Figure 5-15.

Figure 5-15. Data Element Class Diagram



DataElementSet

The DataElementSet is a collection of DataElement objects. Each DataElement object has a name associated with it that is used to refer to the DataElement. DataElementSets also contain a Mutex object that is used to protect the DataElementSet from simultaneous access by multiple DataTransformers or DataRelationships. A list of properties and methods for this class can be found in Table 5-13.

Table 5-13. Properties and Methods for DataElementSet Class

Private Properties	
MElements : map<string, EA_DataElement>	An associative array which contains the set of data elements associated with their names.
Public Methods	
EA_DataElementSet () : EA_DataElementSet	Default constructor.
EA_DataElementSet (names : vector<string>) : EA_DataElementSet	TBD.
GetDataElement (name : const RWCString&) : EA_DataElement*	Return the data element whose name matches the name given.
~EA_DataElementSet () :	Destructor.
AddDataElement (name : const RWCString&) : EA_DataElement*	Creates a new data element with the specified name, adds it to the set, and returns a pointer to it. If the specified name is already in use, returns null.
DeleteDataElement (name : const RWCString&) : void	Removes from the set and deletes the data element with the specified name.
SetDataElement (name : const RWCString&, value : const RWCString&, state : enum EA_State = READY) : void	Sets the value and state of the specified data element. If the data element does not already exist, it is created and added to the DES.
EvaluateState () : enum EA_State	Evaluates the state of the DES, and notifies its observers if the state has changed.
GetNumElements () : size_t	Returns the number of data elements within the DES.

DataElement

The DataElement class represents a chunk of data that can take one of four forms: a scalar value, an array of scalar values, a record of scalar values, or a 2-D table of scalar values in which each column can contain a different data type.Methods are provided for setting and retrieving the information the DataElement. A list of properties and methods for this class can be found in Table 5-14.

Table 5-14. Properties and Methods for DataElement Class

Private Properties	
mNames : vector<string>	The names of each column of the table.
mValues : vector<vector<string>>	The row of values for each column of the table (i.e. a 2-D array).
mLabels : vector<string>	The labels for each column of the table.
mUnits : vector<string>	The units for each column of the table.
mTypes : vector<EA_DataType>	The data types of each column of the table.
mDomains : vector<string>	The domains of each column of the table.
mLimits : vector<string>	The limits of each column of the table.
mFormat : vector<string>	The format of each column of the table.
mNumRows : size_t	The number of rows in the table.
mNumCols : size_t	The number of columns in the table.
mState : EA_State	The current state (set or unset) of the data element.
Public Methods	
EA_DataElement (rows : size_t = 1, cols : size_t = 1, state : enum EA_State = WAITING) : EA_DataElement	Contructor. Creates a data element with the specified number of rows and columns.
EA_DataElement (name : const RWCString&, value : const RWCString&, state : enum EA_State = READY) : EA_DataElement	Constructor. Creates a scalar (i.e. 1x1) data element with the given name and value.
~EA_DataElement () :	Destructor.
getDimension () : enum EA_Dimension	Returns the dimension of the data element. Returns either SCALAR (1x1), ARRAY (Nx1), STRUCT (1xN), or TABLE (NxN).

Table 5-14. Properties and Methods for DataElement Class (Continued)

Private Properties	
NumRows () : size_t numCols () : size_t getState () : enum EA_State getName (col : size_t = 0) : const RWCString& getLabel (col : size_t = 0) : const RWCString& getUnits (col : size_t = 0) : const RWCString& getDomain (col : size_t = 0) : const RWCString& getType (col : size_t = 0) : enum EA_DataType getValue (row : size_t = 0, col : size_t = 0, units : const RWCString& = "") : RWCString getValue (units : const RWCString&) : RWCString getTable () : RWCString getRow (row : size_t = 0) : RWCString getCol (col : size_t = 0, units : const RWCString& = "") : RWCString setName (name : const RWCString&, col : size_t = 0) : void setLabel (label : const RWCString&, col : size_t = 0) : void setUnits (units : const RWCString&, col : size_t = 0) : void setDomain (domain : const RWCString&, col : size_t = 0) : void setType (type : enum EA_DataType, col : size_t = 0) : void setValue (value : const RWCString&, state : enum EA_State = READY) : void setValue (value : const RWCString&, row : size_t, col : size_t) : void setState (state : enum EA_State) : void	Gets or sets the various attributes of the data element.

DataElementIterator

The DataElementIterator class allows programs to iterate through all the elements in a DataElementSet. Because the DataElementIterator is a separate object and maintains its own state, it allows multiple threads to iterate over the same DES simultaneously. A list of properties and methods for this class can be found in Table 5-15.

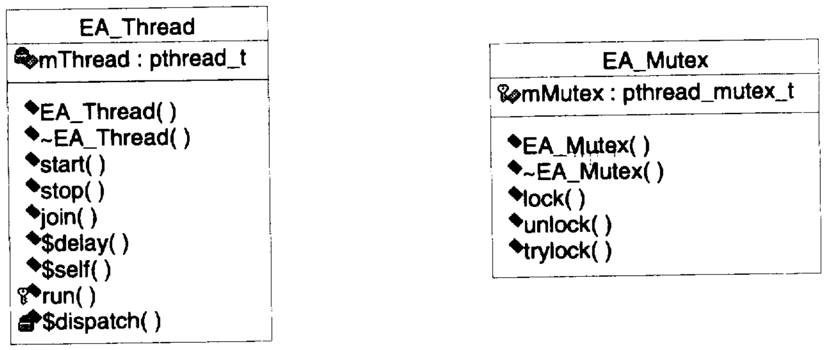
Table 5-15. Properties and Methods for DataElementIterator Class

Private Properties	
mIterator : iterator<string, EA_DataElement>	The underlying iterator that this class wraps.
Public Methods	
EA_DataElementIterator (set : EA_DataElementSet&) : EA_DataElementIterator	Constructor. Creates an iterator to iterate over the elements of the specified DES.
getDataElement () : EA_DataElement*	Returns a pointer to the current data element.
~EA_DataElementIterator () :	Destructor.
reset () : void	Reset the iterator to its initial position and state.
operator ++ () : bool	Advances the position of the iterator and returns true if the new position is valid, false if the end of the set has been reached and the new position is not valid.
getName () : RWCString	Returns the name of the current data element.

THREADS PACKAGE

The Threads package contains a Thread class and a Mutex class. The classes act as object-oriented wrappers for POSIX APIs that support basic multithreading. The Thread class acts as a base class for classes that require a separate thread of execution in a program. The Mutex class provides a locking mechanism for classes that can be used by more than one thread at a time and must ensure that certain operations are executed by only one thread at a time. The class diagram for the Threads package is shown in Figure 5-16.

Figure 5-16. Threads Class Diagram



Thread Class

The Thread class acts as a base class for classes that require a separate thread of execution within a program. The "start" method creates the thread and calls the "run" method that is provided by the subclass to act as the body of the thread. The "stop" method cancels execution of the thread and the "join" method provides a synchronization mechanism by waiting (i.e., blocking the caller) until the thread has finished. A list of properties and methods for this class can be found in Table 5-16.

Table 5-16. Properties and Methods for Thread Class

Private Properties	
mThread : pthread_t	The id of the underlying POSIX thread.
Public Methods	
EA_Thread () : EA_Thread	Constructor.
~EA_Thread () :	Destructor.
start () : void	Starts the thread and calls the "run" method defined by the subclass.
stop () : void	Cancels (aborts) execution of the thread.
join () : void	Waits for the thread to finish execution.
delay (secs : unsigned int) : void	Pauses the specified number of seconds.
self () : pthread_t	Returns the thread id of the thread which calls the function.
Protected Methods	
run () : void	A virtual function which must be defined by the subclass to be the body of the thread.

Mutex Class

The Mutex class provides a exclusive locking mechanism that allows only one thread at a time to execute a "critical" section of code. The Mutex class typically is used by classes that act as shared communication mechanisms between two or more threads. One thread "owns" the mutex at any given time. The "lock" mechanism waits until the current owner is finished, which it signals by calling the "unlock" mechanism. The "trylock" method is similar to "lock", but instead of waiting until the current owner is done it returns false if the mutex is unavailable. A list of properties and methods for this class can be found in Table 5-17.

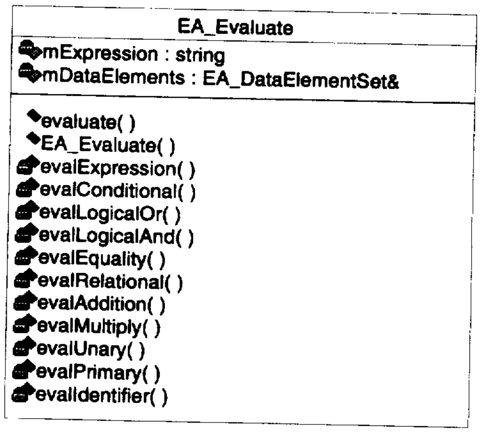
Table 5-17. Properties and Methods for Mutex Class

Protected Properties	
mMutex : pthread_mutex_t	The id of the underlying POSIX mutex.
Public Methods	
EA_Mutex () : EA_Mutex	Constructor.
~EA_Mutex () :	Destructor.
lock () : void	Waits the current owner of the mutex (if any) to finish, then locks the mutex (i.e. claims ownership).
unlock () : void	Unlocks the mutex and allows a waiting thread to lock it and continue execution.
trylock () : bool	If the thread is unlocked (i.e. no other thread is currently using it), locks the thread and returns true. Otherwise returns false.

UTILITY PACKAGE

The Utility package contains the Evaluate class, which is used by the DataRelationship class to evaluate the expressions that make up a data relationship specification. The class diagram for the Utility package is shown in Figure 5-17.

Figure 5-17. Utility Class Diagram



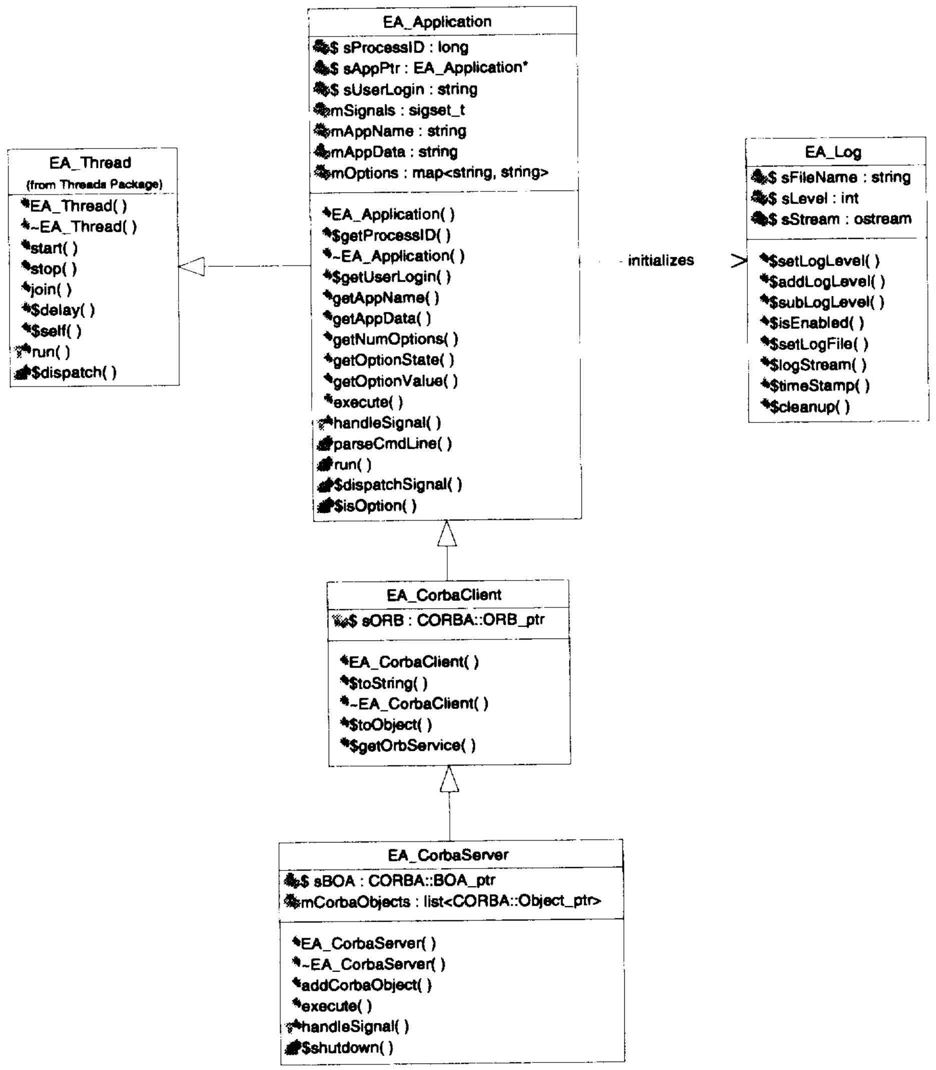
Evaluate Class

The Evaluate class is used to evaluate C-style mathematical expressions, which are used in data relationship specifications. The "evaluate" method returns the numeric value of the expression. The input DES of the data relationship is used to look up the values of variables that occur in the expression.

APPLICATION PACKAGE

The Application package contains classes that encapsulate the basic behavior of CORBA client and server applications. The package also contains an Application class that is a parent class of the CORBA client & server classes. The Application class encapsulates basic application-level behavior, such as signal handling, command-line parsing, and error logging. The class diagram for the Application package is shown in Figure 5-18.

Figure 5-18. Application Class Diagram



Application Class

The Application class provides basic application-level functions, such as signal handling, command-line, parsing, and initialization of the application error log. It has a separate thread (by inheriting from the Thread class) which provides handling of asynchronous signals. Both synchronous and async signals are dispatched to the "handleSignal" method, which can be overridden by subclasses. Various methods exist for getting command-line information, such as whether an option was given or not, or the value of an option that takes a parameter. The Application Class also defines an "execute" method that must be provided by the subclass to define the body of the application. A list of properties and methods for this class can be found in Table 5-18.

Table 5-18. Properties and Methods for Application Class

Private Properties	
sProcessID : long	The process id of the current executable.
sAppPtr : EA_Application*	Static variable which points to the single application object in the program.
sUserLogin : string	The login (i.e. id) of the user that ran the current executable.
mSignals : sigset_t	The set of asynchronous signals which the application handles.
mAppName : string	The file name of the current executable program (i.e. argv[0]).
mAppData : string	The data which was passed to the application on the command line.
mOptions : map<string, string>	The list of options and arguments which were passed to the application on the command line.
Public Methods	
EA_Application (argc : int, argv : char**) : EA_Application	Constructor. Parses the command line, registers the signal handlers, initializes the error log, and starts the signal handling thread.
getProcessID () : long	Returns the process id of the current executable.
~EA_Application () :	Destructor. Stops the signal handling thread, and cleans up the log file if necessary.
getUserLogin () : RWCString&	Returns the user login of the user that executed the application.
getAppName () : const RWCString&	Returns the name of the executable (i.e. argv[0]).
getAppData () : const RWCString&	Returns any data passed to the application on the command line.
getNumOptions () : size_t	Gets the number of options passed to the application on the command line.
getOptionState (option : const RWCString&) : bool	Returns true if the specified option was passed to the application on the command line.

Table 5-18. Properties and Methods for Application Class (Continued)

execute () : void	A virtual function which must be defined by the subclass to contain the body of the application.
Protected Methods	
handleSignal (signal : int) : void	A virtual function which provides basic signal handling capabilities. Can be overridden by the subclass.

Log Class

This class provides error and debug message logging for an application. The log messages can be written to standard output (the default) or a user-specified file. Debug levels can be set that allow selective filtering of debug information at run-time. A number of macros also are provided to simplify calling the "log Stream" method. A list of properties and methods for this class can be found in Table 5-19.

Table 5-19. Properties and Methods for Log Class

Private Properties	
sFileName : string	The name of the log file (if any) to send messages to.
sLevel : int	The current log level, which decides the level (i.e. detail) of messages to send to the log.
sStream : ostream	The current output stream that log messages are sent to.
Public Methods	
setLogLevel (level : int) : void	Sets the current log level to the specified value.
addLogLevel (level : int = 1) : void	Increments the log level by the specified value.
subLogLevel (level : int = 1) : void	Decrements the log level by the specified value.
isEnabled (level : int) : bool	Returns whether a message of the specified level should be logged. Returns true if the specified level is less-than-or-equal to the current log level.
setLogFile (logFile : string) : void	Creates and sets the log file to a file with the specified name. All subsequent log messages will be sent to this file.
logStream (line : unsigned int, file : string, tag : char) : ostream&	Writes the given line number, file name, and tag to the log along with a time stamp, then returns the log stream which can be used to output a log message.
timeStamp () : string	Returns a string which contains the current date and time.
cleanup () : void	Performs cleanup by closing the log file if necessary.

CorbaClient Class

The CorbaClient class is a subclass of the Application class that handles initialization of the ORB as well as encapsulating other basic CORBA client functions, such as object-to-string and string-to-object conversion. A list of properties and methods for this class can be found in Table 5-20.

Table 5-20. Properties and Methods for CorbaClient Class

Protected Properties	
sORB : CORBA::ORB_ptr	A reference to the ORB being used by the application.
Public Methods	
EA_CorbaClient (argc : int, argv : char**) : EA_CorbaClient	Constructor. Initializes the ORB.
toString (objRef : CORBA::Object_ptr) : RWCString	Converts a CORBA object reference into a string representation.
~EA_CorbaClient () :	Destructor.
toObject (strObj : const char*) : CORBA::Object_ptr	Converts a string representation of an object reference (a "stringified" object reference) into an object reference.
getOrbService (name : const char*) : CORBA::Object_ptr	Returns a reference to the given ORB service registersed with the ORB.

CorbaServer Class

The CorbaServer class handles initialization of the BOA (Basic Object Adapter) and encapsulates the registration and activation of CORBA objects. The class also provides a signal handler that provides a clean shutdown of the server in the event of certain external signals or internal error signals. A list of properties and methods for this class can be found in Table 5-21.

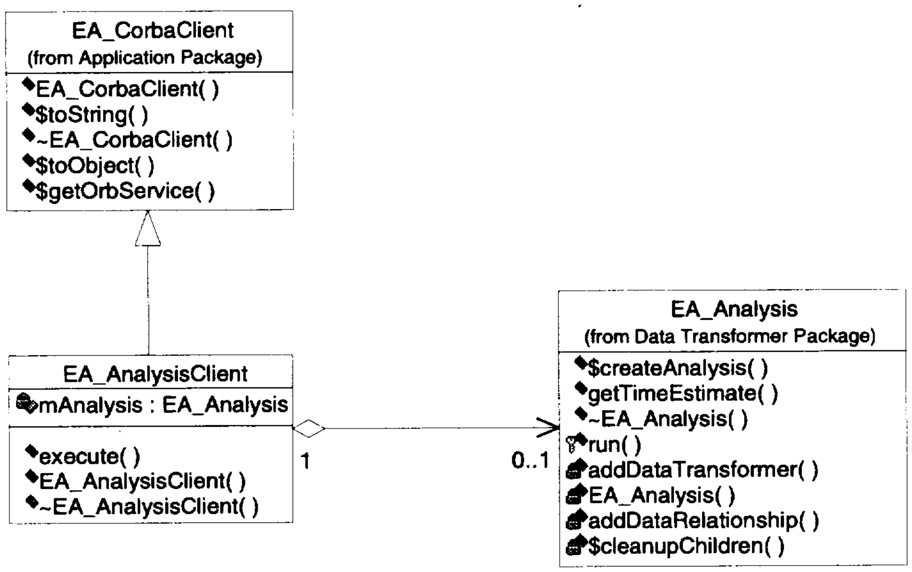
Table 5-21. Properties and Methods for CorbaServer Class

Private Properties	
sBOA : CORBA::BOA_ptr	A reference to the BOA (Basic Object Adaptor) used by the server.
mCorbaObjects : list<CORBA::Object_ptr>	A list of CORBA objects which the server contains & manages.
Public Methods	
EA_CorbaServer (argc : int, argv : char**) : EA_CorbaServer	Constructor. Initializes the BOA.
~EA_CorbaServer () :	Destructor. Deactivates and releases the CORBA objects owned by the server.
addCorbaObject (objRef : CORBA::Object_ptr) : void	Registers and activates the specified CORBA object, and adds it to the list of CORBA objects owned by the server.
execute () : void	Executes the CORBA event loop which continually accepts incoming requests and passes them to the proper object to be handled.
Protected Methods	
handleSignal (signal : int) : void	Overrides the Application class signal handling to properly shut down the CORBA server when certain signals are received.

ANALYSIS CLIENT PACKAGE

The Analysis Client package contains the driver for the AnalsysiClient application. The class diagram for the Analysis Client package is shown in Figure 5-19.

Figure 5-19. Analysis Client Class Diagram



AnalysisClient Class

The AnalysisClient class inherits from the CorbaClient class and provides the driver for the POC client application. The "execute" method instantiates an Analysis, runs it, and then displays the results when it has finished. A list of properties and methods for this class can be found in Table 5-22.

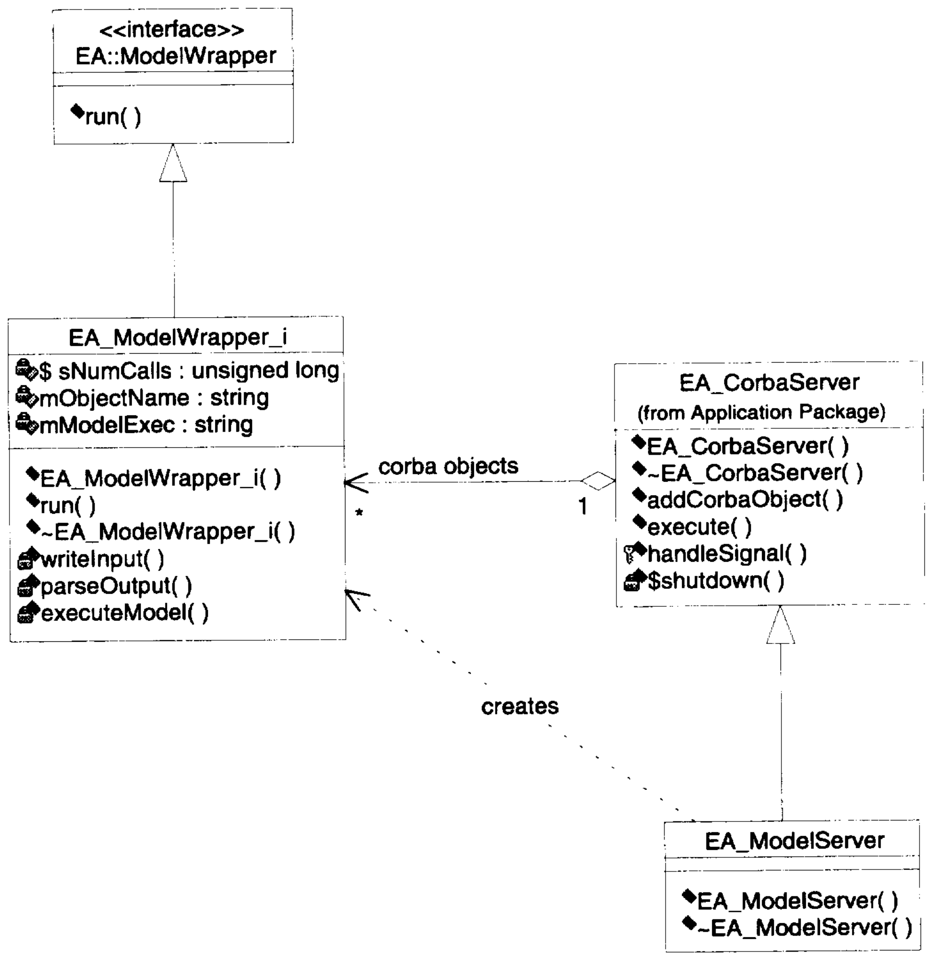
Table 5-22. Properties and Methods for AnalysisClient Class

Private Properties	
mAnalysis : EA_Analysis	The analysis which the analysis client runs.
Public Methods	
execute () : void	Creates an analysis based on the specification given on the command line, prompts for any necessary input, then runs the analysis and outputs results.
EA_AnalysisClient () : EA_AnalysisClient	Constructor.
~EA_AnalysisClient () :	Destructor.

MODEL SERVER PACKAGE

The Model Server package contains all the classes that are used to implement the CORBA server application for the POC. The package includes the generic Interface definition language (IDL) for models, the ModelWrapper_i class that implements the IDL interface, and the class that acts as the driver for the server. The class diagram for the Model Server package is shown in Figure 5-20.

Figure 5-20. Model Server Class Diagram



ModelWrapper Interface

The ModelWrapper interface is an IDL that provides a standard, generic interface to distributed models in the ASAC EA system. The ModelWrapper interface provides a "run" method that takes a sequence of data as input and returns a sequence of data as output.

ModelWrapper_i Class

The ModelWrapper_i class provides an implementation of the ModelWrapper interface that wraps standalone models with simple file-oriented interfaces. The class parses the input sequence, writes it to a file, and executes the model against the input file. It then parses the models output file and returns the results as a sequence of data.

ModelServer Class

The ModelServer class inherits from the CorbaServer class and provides the driver for the POC server application. The ModelServer class reads from a configuration file specified on the command line and creates one or more ModelWrapper_i objects and registers and activates them as CORBA objects. Then it runs as a server, accepting and handling requests sent to those objects, until it is shutdown or otherwise killed.

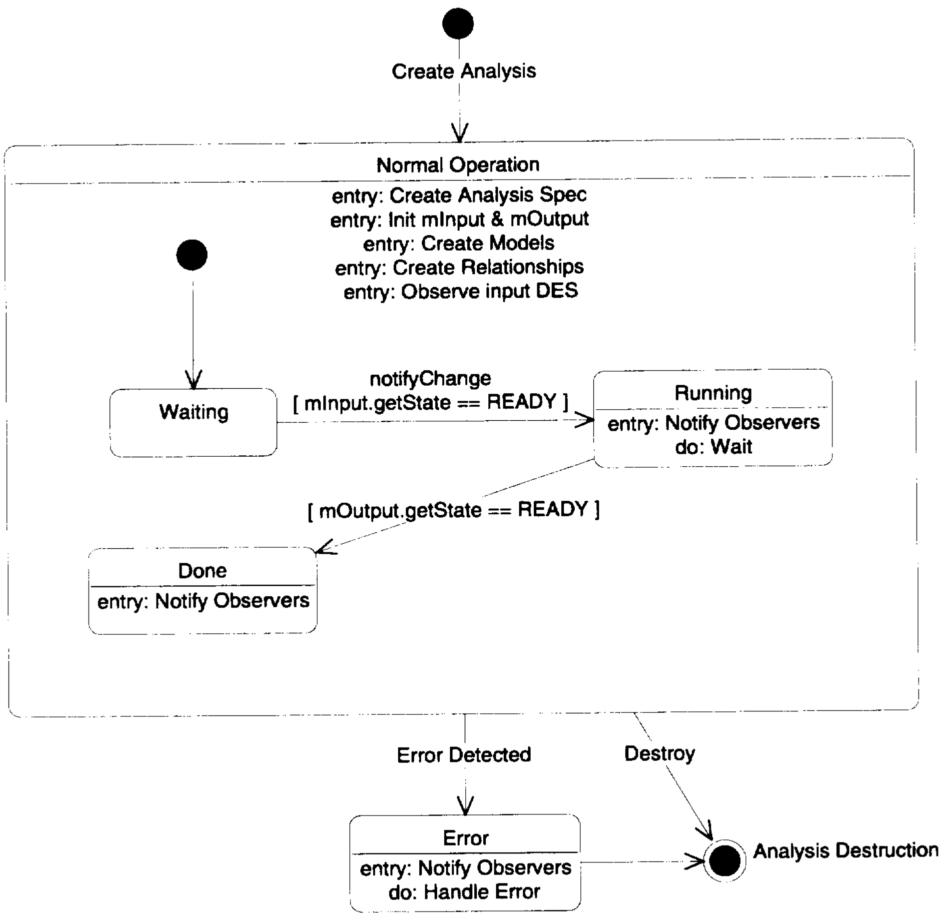
Domain-Specific Software Architecture Substage 4-7: Develop State Diagrams

State diagrams describe all possible states of a particular object and how the object's state changes on particular events. The following sections contain state diagrams only for the classes that require states.

ANALYSIS STATE DIAGRAM

The Analysis has four states: "Waiting," "Running," "Done," and "Error." On creation of the Analysis, it creates an AnalysisSpecification, receives input and output DataElementSets from the AnalysisSpecification, and registers as an observer to its input DataElementSet. At the same time, the initial state of the Analysis is set to the "Waiting" state. When the Analysis is notified of a state change on its input DataElementSet, the Analysis finds out what the state is. If the input DataElementSet is in the "Set" state, the Analysis will change to the "Running" state and notify its observers. In the "Running" state, it will create the Models and DataRelationships needed for the Analysis and will wait for its output DataElementSet to become "Set." Once the output DataElementSet of the Analysis is "Set," it will go to the "Done" state where it will remain until either the Analysis input DataElementSet becomes "Unset" (at which time the Analysis will go to the "Not Ready To Run" state), or it is destroyed. Upon a system error, the Analysis will go to the "Error" state. Figure 5-21 shows the Analysis state diagram.

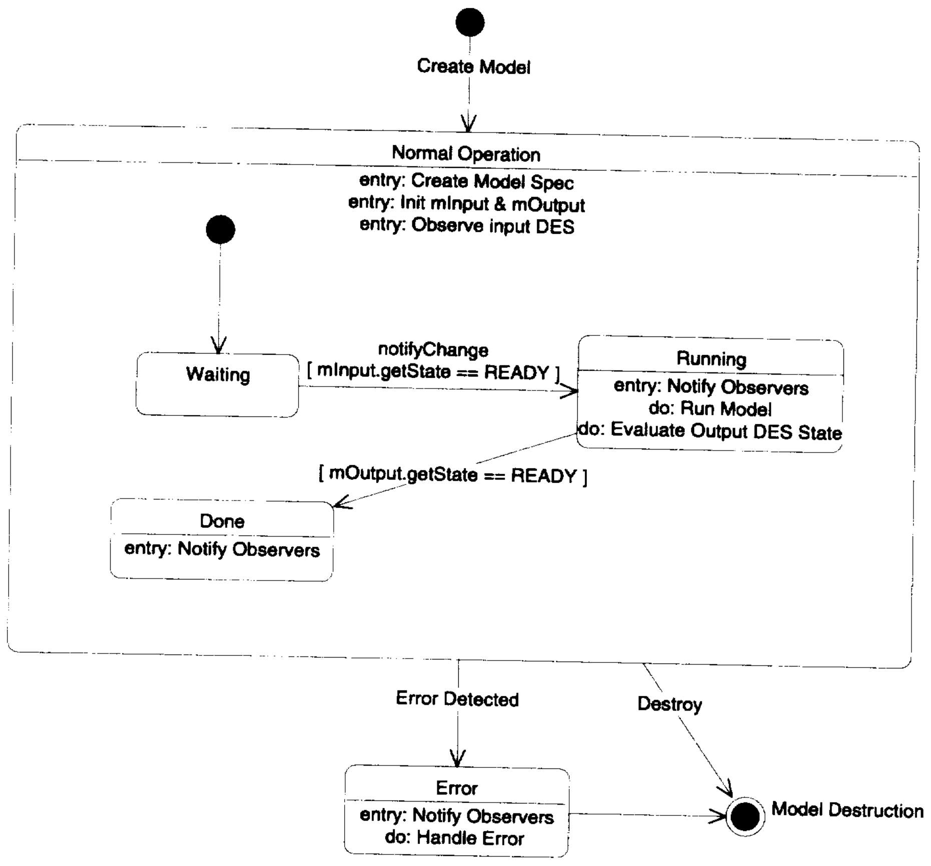
Figure 5-21. Analysis State Diagram



MODEL STATE DIAGRAM

The Model class has four states: "Waiting," "Running," "Done," and "Error." On creation of the Model, it creates a ModelSpecification, receives input and output DataElementSets from the ModelSpecification, and registers as an observer to the input DataElementSet. At the same time, the initial state of the Model is the "Waiting" state. When the Model's input DataElementSet goes to the "Set" state, the Model will change state to the "Running" state and notify its observers. In the "Running" state, the Model will perform it's transformation and will wait for its output DataElementSet to become "Set." Once the Model output DataElementSet is "Set," it will go to the "Done" state and will remain there until either the Model input DataElementSet becomes "Unset" (at which time the Model will go to the "Waiting" state), or it is destroyed. Upon a system error, the Model will go into the "Error" state. Figure 5-22 shows the Model state diagram.

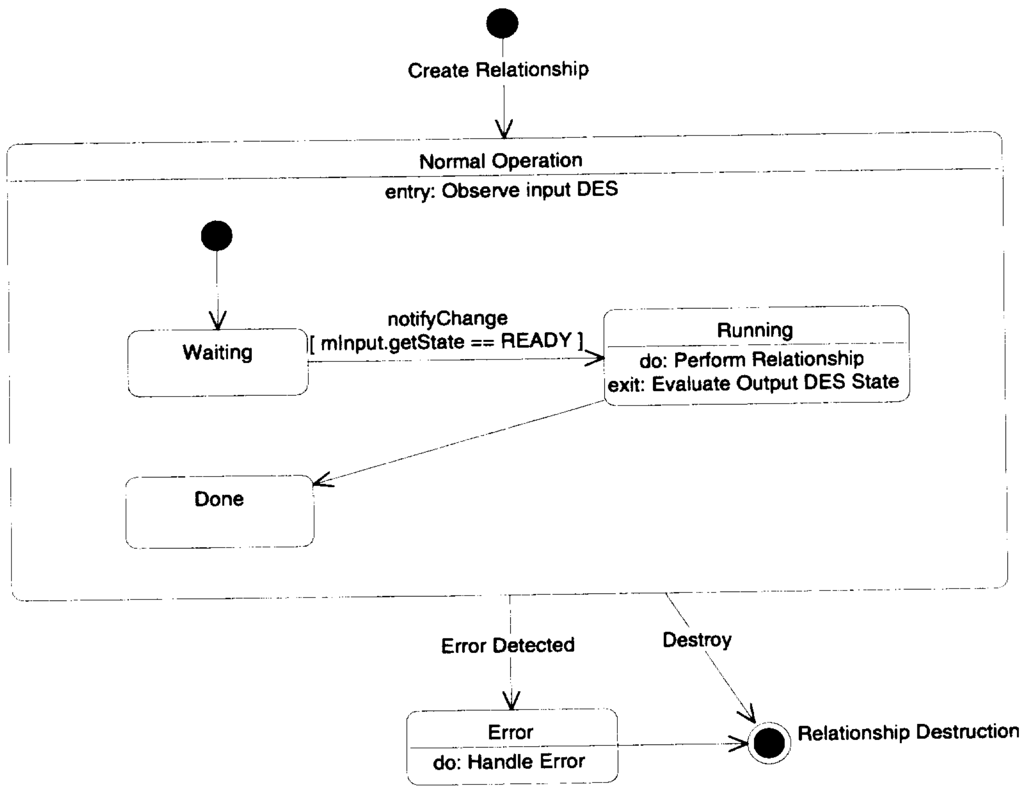
Figure 5-22. Model State Diagram



DATARELATIONSHIP STATE DIAGRAM

The DataRelationship class has four states: "Waiting," "Running," "Done," and "Error." On creation of the DataRelationship, it creates the DataRelationship Specification. The DataRelationship then receives input and output DataElement-Sets from the DataRelationship Specification and registers as an observer to the input DataElementSet. At the same time, the initial state of the DataRelationship is the "Waiting" state. When its input DataElementSet goes to the "Set" state, the DataRelationship will change state to the "Running" state and notify its observers. In the "Running" state, the DataRelationship will perform its transformation and will wait for its output DataElementSet to become "Set." Once the DataRelation-ship output DataElementSet is "Set," it will go to the "Done" state and will re-main there until either the DataRelationship input DataElementSet becomes "Unset" (at which time the Model will go to the "Waiting" state), or it is de-stroyed. Figure 5-23 shows the DataRelationship state diagram.

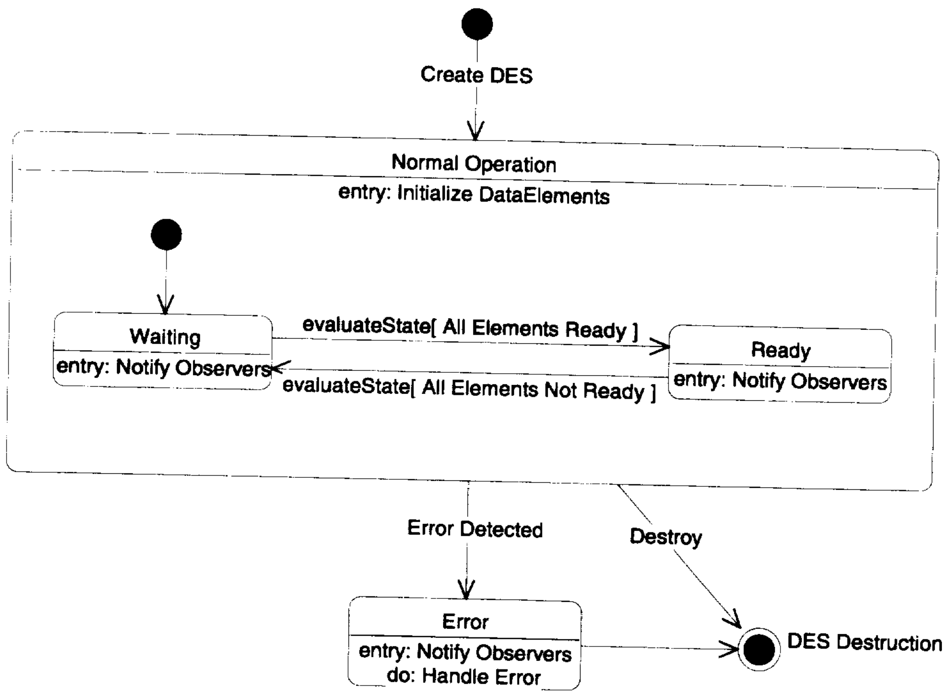
Figure 5-23. DataRelationship State Diagram



DATAELEMENTSET STATE DIAGRAM

The DataElementSet class has three states: "Waiting", "Ready," and "Error." On creation of the DataElementSet, its initial state will be the "Waiting" state. The DataElementSet will go to the "Ready" state when it evaluates its state and finds all of its DataElements are in the "Ready" state. Upon a system error, DataElementSet will go into the "Error" state. Figure.5-24 shows the DataElementSet state diagram.

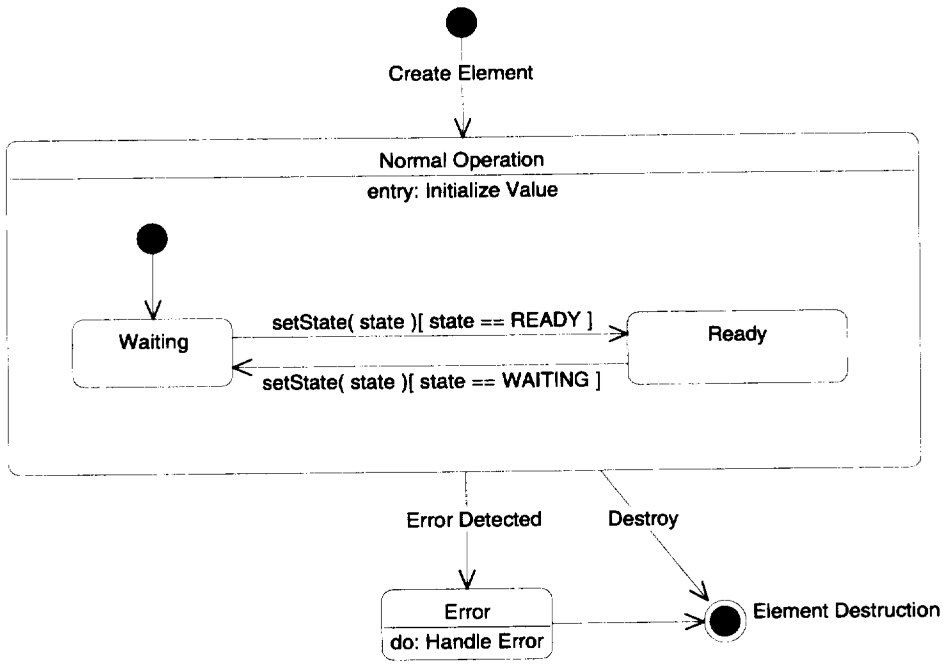
Figure 5-24. DataElementSet State Diagram



DataElement State Diagram

The DataElement class has three states: "Waiting", "Ready" and "Error." On creation of the DataElement, its initial state will be the "Waiting" state. The DataElementSet will go to the "Ready" state when its state is changed to "Ready" by the SetState() command. It will change to the "Waiting" state when the SetState() command sets it to "Waiting." Upon a system error, DataElement will go into the "Error" state. Figure 5-25 shows the DataElement state diagram.

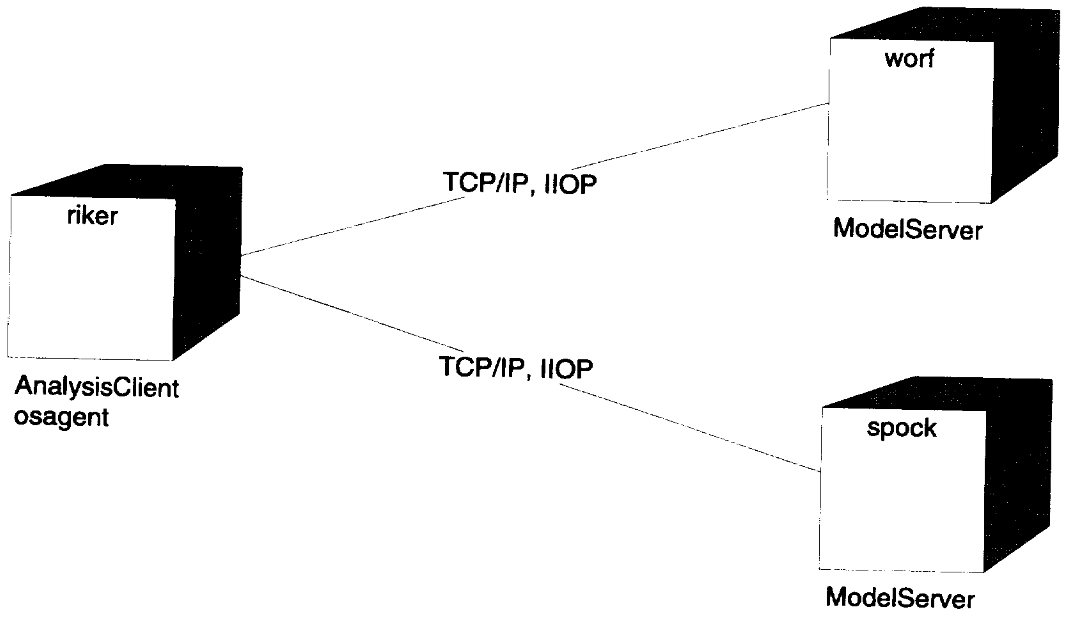
Figure 5-25. DataElement State Diagram



DSSA Substage 4-8: Develop Deployment Diagrams

A deployment diagram shows processors, devices, and their connections. A processor is a hardware component capable of executing programs, i.e., a computer. A device is a hardware component with no computing power, i.e., hardware controller or modem. There are no devices in the ASAC EA system, so the POC Deployment Diagram contains only processors and their connections with each other. The Deployment Diagram is shown in Figure 5-26. It is a generic model showing that there will be an AnalysisClient, riker, with an osagent on it as well as multiple ModelServers, spock and worf, that will have Model Applications running on them.

Figure 5-26. POC Deployment Diagram



DSSA Substage 4-9: Review and Iterate

Review and iterate the items developed in DSSA stage 4.

DSSA Stage 5—Identify Reusable Artifacts

The goal for this phase of the domain-engineering process is to populate the software architecture high-level design(s) with components that may be used to generate new applications in the domain.

The following substages of DSSA stage 5 were completed during the ASAC development effort:

- 5-1 Develop and collect the reusable artifacts
- 5-2 Develop each module
- 5-3 Requirements, verification, and testing
- 5-4 Review and iterate.

DSSA Substage 5-1: Develop and Collect the Reusable Artifacts

This substage addresses how to determine the best source of components to populate the software architecture. It is often referred to as the make, buy, or modify decision.

ASAC SERVICES

Message broker and binding languages evaluated and selections during the design effort and documented in the *Aviation System Analysis Capability Executive Assistant Design*. During development, we evaluated the need for an object-oriented database management system, that uses the same techniques described in the design document.

The ASAC EA system needs to store and retrieve data related to the execution of models. A data management service was identified in the system architecture to provide this functionality. A number of options were available for providing the service, but to provide a robust and scaleable solution, we decided to use a commercial database management system.

Many types of database management systems exist. Four main categories of systems are relational, object-oriented, object-relational, and hierarchical. The last two categories of databases are primarily used in specialized applications that do not match the domain of the ASAC system. Therefore, the first two categories were investigated to provide the data management service.

Relational databases, e.g., Sybase and Oracle, are the most commonly used databases, and are widely used in business applications. In these databases, data are stored in tables consisting of rows and columns of data; much like a simple Excel spreadsheet. Despite their name, relational databases can become quite complex and unwieldy when complex relationships between tables exist. Another problem with relational databases is that their model does not match the object-oriented paradigm, requiring writing additional layers of code to handle mappings between objects and the relational database. The extra layer of code can be quite complex and require a great deal of debugging.

Object-oriented databases are relatively new, but have quickly gained acceptance in certain domains as the products have matured. Unlike relational databases, object-oriented databases store objects and their relationships directly, making storing data and translating between objects in the code and the database almost seamless. In applications that are highly object-oriented and involve complex relationships, like the ASAC EA, an object-oriented database can save a great deal of time in developing and maintaining the system. In addition, an object-oriented database can be orders of magnitude faster in these types of applications because they store relationships directly, thereby avoiding the need to perform complex relational joins. For these reasons, we decided that an object-oriented database would be the best choice for the data management service.

A number of object-oriented database management systems (OODBMS) were initially investigated. They included

- GemStone
- O2 (Ardent Software)
- Objectivity
- ObjectStore (ObjectDesign)
- POET
- Versant.

On the basis of our initial research and general selection criteria, such as the platforms and languages supported, three databases were selected for more thorough evaluation. They are

- O2 (Ardent Software)
- ObjectStore (ObjectDesign)
- Versant.

We formulated evaluation criteria and questions to use as guidelines and areas of investigation during the detailed evaluation phase. Unlike relational databases, OODBMSs differ from one another significantly. Choosing the best one depends greatly on the specific application and requirements. To assess the differences, we contacted and questioned technical representatives of the three vendors. Also, each representative supplied evaluation copies of their OODBMS.

We tested the evaluation software to determine, at a minimum

- Ease of installation
- Ease of administration
- Basic functionality (using included demonstration programs)
- Ease of porting existing ASAC EA code.

We chose Versant because it was the only database that performed acceptably in all the evaluation areas. O2 did not enable us to easily port existing ASAC EA code and did not include a suitable persistent collection class library. ObjectStore proved difficult to install and administer and was a clumsier overall interface than the other two products.

DSSA Substage 5-2: Develop Each Module

DEVELOPMENT ENVIRONMENT

The environment used for developing the ASAC EA POC consisted of desktop and server machines, as well as a number of tools and libraries which are described below. A diagram of how the machines, tools, and libraries were configured is in Figure 5-27.

- Machines

  HP 9000/803 Server running HP-UX (unix) version 10.20
  The HP server machine was used for developing and testing all the C++ code for the POC.

  Compaq Desktop PCs running Windows 95
  Desktops PCs were used for initial prototyping of the design in the Java language and for reverse engineering the design when the POC was completed. The PCs also served as an interface to the server machine where the actual development took place.

- Tools and Libraries

  HP C++ Compiler (aC++)
  The HP C++ compiler was used to compile and debug the C++ code developed for the POC.

  VisiBroker for C++
  VisiBroker for C++ is the CORBA Object Request Broker used to build and deploy the POC in a distributed environment across multiple platforms.

  Revision Control System
  Revision Control System (RCS) is a “revision control” or “version control” system the was used to baseline the code at the end of the POC development phase. All development done during the next phase of development will use that code as a baseline and RCS will be used to track and manage any changes made to the code.

  make
  Build scripts were written on the server using a utility called make. The scripts provide automated builds (compilation) of the developed source code.

Rational Rose for C++
Rational Rose was used during development to view the existing design. It also was used after development to reverse-engineer and update the design based on the developed code.

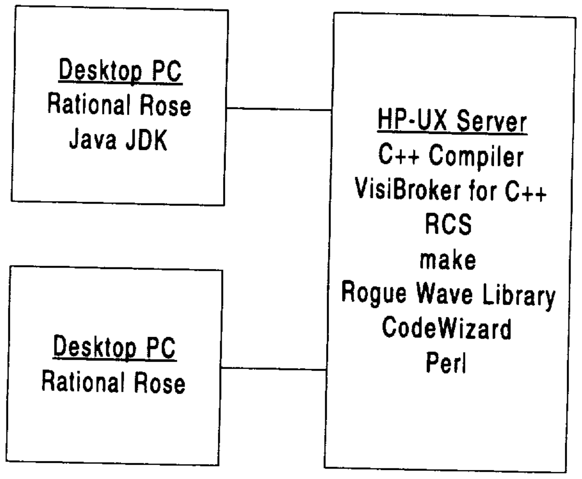
RogueWave Tools.h++ library
The RogueWave Tools library provides a number of utility and container classes that were used in the development of the C++ code for the POC.

Parasoft CodeWizard
CodeWizard finds programming and design problems in C++ or Java source code automatically.

Java JDK 1.1.5
The Java Development Kit from Sun was used for developing a quick prototype of the POC before beginning development in C++.

Perl 5.004
The Practical Extraction and Report Language (perl) is a concise general-purpose language often used for scanning text and printing formatted reports. Its Common Gateway Interface (CGI) and libraries make it well suited for forms processing and on-the-fly page creation. Perl also contains object-oriented features.

Figure 5-27. ASAC EA Proof of Concept Development Environment



Development Process

The development started with the design that was completed during the previous phase of the project and documented in the *Aviation System Analysis Capability Executive Assistant Design.* The development steps were

- Prototyping

- Requirements Definition
- Coding and Unit Testing
- Integration
- System Testing
- Documentation.

A prototype was developed initially for verifying key aspects of the design and for discovering implementation issues early in the development. The prototype was developed in the Java language on desktop PCs. Java was chosen because it allows rapid development, has a CORBA binding, has built-in support for threads, and maps very well to C++. Also when the prototype was being developed, the IDL interface for the CORBA ModelServer was designed and tested.

Requirements then were defined in detail based on information from the architecture and design documents. Nine categories of requirements were developed which contain a total of sixty-two detailed requirements.

After the prototype was finished and requirements were carefully defined, the full system was coded in C++. Classes were coded according to the design and the *C++ Coding Standards* chosen for the project. When necessary, test drivers were written to test individual classes or a group of classes. Additional classes were designed and implemented as required for encapsulating certain functionality, e.g., threads and error logging. As issues arose during the development, solutions were prototyped or tested as necessary and then implemented. As classes were finished, they were put under configuration management using RCS.

When the core functionality of the classes had been completed, the classes were integrated into the two pieces of the POC: the AnalysisClient and the ModelServer. Each was integrated as early as possible to avoid redesign and rework caused by undiscovered problems propagating to the end of the development phase. As additional functionality was developed, it was integrated into the system.

System-level test procedures were developed based on the basis of the functional requirements of the system. Once developed, the system was run against the test cases and the results verified. Deficiencies were logged as a problem report and tracked until the problem was corrected and re-verified.

Once the POC was completely developed, its complete design was captured to use as input for the design and development of the beta system. One possible method to capture the design was to modify the existing design, making changes and adding additional classes as necessary. However, Rational Rose, the design tool used, provided a slightly more elegant solution. The finished code was analyzed by Rational Rose, and the class model was reverse-engineered from the actual code. This process is not perfect and the resulting model had to be modified, but much less than would have been required for the existing model.

Bank Prototype

A banking demonstration or example program that shipped with the Visigenic ORB was used as the basis for a quick prototype to test some of the concepts relating to the ASAC EA client-server communication. Three main areas were tested

1. Communications over CORBA between a Java client and a C++ server
2. Asynchronous callbacks from the server to the client
3. Execution of the Java client as an applet running in a browser.

Versions of the Visigenic program were available in both Java and C++, which enabled us to easily test the first area, communication between a Java client and a C++ server.

The second area was tested by modifying interfaces between the client and server to allow the server to call CORBA objects on the client, which in turn updated the client. This area was tested without problems.

The third area was tested by modifying the client to run as an applet in a browser. Although this area is more trivial than the first two areas, it was the most problematic because of inaccurate and insufficient documentation in the Visigenic manuals. Solutions to the problems were discovered only by searching various CORBA-related Internet newsgroups. Once the problems were remedied, the area was successfully completed.

In addition to testing the three areas, the prototype helped to validate aspects of the system design and provided useful information for the implementation and deployment phases of the project.

DSSA Substage 5-3: Requirements, Verification, and Testing

ASAC EA POC Requirements

Fifteen requirements were applicable to the ASAC EA POC. They are:

- AE0001 The Analyst shall have the capability to execute an analysis if an off-line administrator has granted the appropriate permissions.
- AE0003 When an analysis is executed, the names of the models that are executed, as part of that analysis, will be logged to a log file.
- AE0004 When an analysis is executed, its inputs and outputs will be logged.

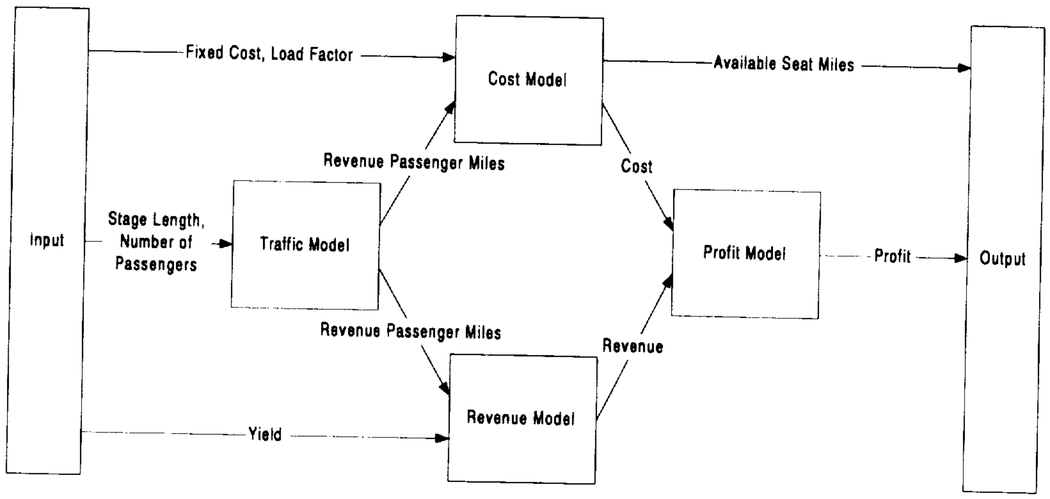
- AE0005 When a model is executed, its inputs and outputs will be logged.
- AE0006 Upon completion of the execution of an analysis, the results will be presented to the user if the user is logged into the system.
- AE0008 ASAC will provide a message to the user indicating a rough estimated time required to execute an analysis. Note: This will be a very rough estimate, as there are currently no plans to perform an interrogation of network and system(s) loading at the time of execution to provide a better estimate, not to mention the affect of data set size on model execution time.
- AM0001 The capability shall be provided to create an analysis by using off-line tools.
- AM0003 The capability shall be provided to update an analysis by using off-line tools.
- AS0001 An analysis may contain one or more models or analyses.
- AS0002 Analyses may have default input values.
- DC0001 ASAC will accommodate operation of its models at remote sites.
- DC0003 ASAC EA shall support the concurrent execution of more than one instance of the same analysis on the same or different machines.
- DC0004 ASAC EA shall support the concurrent execution of more than one instance of the same model on the same or different machines.
- DC0005 The physical location of the models shall be transparent to the ASAC EA.
- EH0003 The user shall be notified if a model server is not available.

These fifteen requirements were validated as part of the ASAC EA POC acceptance.

POC Implementation

Figure 5-28 shows the POC implementation. The POC analysis contains four models, Traffic, Cost, Revenue and Profit; six data relationships; and five user inputs. We chose this configuration because it exercises many of the characteristics of an analysis (i.e., single and multiple data relationships between models, single models feeding multiple models, and multiple models feeding single models).

Figure 5-28. Proof of Concept Implementation



Model inputs, outputs, and calculations are described in Table 5-23 below.

Table 5-23. Proof of Concept Model Descriptions

Model	Inputs	Outputs	Calculations
Traffic	Stage Length Number of Passengers	Revenue Passenger Miles	Revenue Passenger Miles = (Passengers) × (Stage Length)
Cost	Fixed Cost Load Factor	Available Seat Miles Cost	Available Seat Miles = (Revenue Passenger Miles) ÷ (Load Factor) Cost = (Fixed Cost) + (Available Seat Miles) × (0.1)
Revenue	Yield Revenue Passenger Miles	Revenue	Revenue = (Revenue Passenger Miles) × (Yield)
Profit	Cost Revenue	Profit	Profit = (Revenue) - (Cost)

To complete the analysis, the ASAC EA POC performed the following:

- Built the analysis and gave it an input.
- Constructed the models and data relationships between the models and analysis.

- The models transformed their input data into output data.
- When all transformations were finished, and the analysis was complete, stored the final output.

The models used for the analysis were distributed. The analysis communicated with the distributed models using Visigenic's implementation of the OMG CORBA standard that makes the distributed nature of the models virtually transparent. The actual models were wrapped by a standard interface, defined by using OMG IDL that allowed the models to be distributed and provided clients with a standard method for invoking all models. Models for the analysis were developed using perl, with an interface similar to the interface of the current ASAC models. Each model was wrapped to enable distributed communication. The analysis application and the model wrappers were developed on the HP-UX platform by using the C++ programming language. The input and output data structures and default values for each model were specified by a prototype system catalog.

ASAC EA POC Testing

Six procedures were developed to test the fifteen ASAC EA POC requirements. Procedures TP-AE-1 and TP-AE-2 test the Analysis Execution requirements, TP-AM-1 the Analysis Management requirements, TP-AS-1 the Analysis Specification requirements, TP-DC-1 the Distributed Computing requirements, and TP-EH-3 the Error Handling requirement. Table 5-24 maps ASAC EA POC requirements to the appropriate test procedure and lists the implementation classes used to test a requirement.

Table 5-24. ASAC EA Proof of Concept Requirements and Test Procedures

Requirement	Test Procedure	Implementation Classes
AE0001	TP-AE-1	EA_AnalysisClient
AE0003	TP-AE-2	EA_Log, EA_Analysis
AE0004	TP-AE-2	EA_Log, EA_Analysis
AE0005	TP-AE-2	EA_Log, EA_ModelProxy
AE0006	TP-AE-1	EA_AnalysisClient, EA_DataElementSet
AE0008	TP-AE-1	EA_AnalysisClient, EA_AnalysisSpec
AM0001	TP-AM-1	EA_Specification, EA_DataStorage
AM0003	TP-AM-1	EA_Specification, EA_DataStorage
AS0001	TP-AS-1	EA_Analysis, EA_AnalysisSpec
AS0002	TP-AS-1	EA_AnalysisSpec, EA_DataElementSet, EA_DataElement
DC0001	TP-DC-1	EA_ModelProxy, EA_ModelWrapper_i, EA_ModelServer, CORBA

Table 5-24. *ASAC EA Proof of Concept Requirements and Test Procedures (Continued)*

Requirement	Test Procedure	Implementation Classes
DC0003	TP-DC-1	EA_ModelProxy, EA_ModelWrapper_i, EA_AnalysisClient, CORBA
DC0004	TP-DC-1	EA_ModelProxy, EA_ModelWrapper_i, EA_ModelServer, CORBA
DC0005	TP-DC-1	EA_ModelProxy, EA_ModelSpec, CORBA
EH0003	TP-EH-3	EA_ModelProxy

The ASAC EA POC test procedures were successfully completed on 23 February 1998. One minor problem report was issued during the testing. The problem was resolved and rechecked before the test was completed. A summary of the test results is in Table 5-25. The as-run test procedures are in Appendix A.

Table 5-25. *Summary of the ASAC EA Proof of Concept Test Results*

Test #	Date	Tested by	Witnessed by	Results
TP-AE-1	February 23, 1998	Kevin Anderson	Eileen Roberts	Passed
TP-AE-2	February 23, 1998	Kevin Anderson	Eileen Roberts	Passed
TP-AM-1	February 23, 1998	Kevin Anderson	Eileen Roberts	PR #1 Issued, Passed
TP-DC-1	February 23, 1998	Kevin Anderson	Eileen Roberts	Passed
TP-AS-1	February 23, 1998	Kevin Anderson	Eileen Roberts	Passed
TP-EH-3	February 23, 1998	Kevin Anderson	Eileen Roberts	Passed

ASAC EA POC DEMONSTRATION

After the ASAC EA POC was tested successfully, it was demonstrated to NASA. Four scenarios were demonstrated

- Single Analysis
- Multiple Analysis
- Load Balancing
- Fault Tolerance.

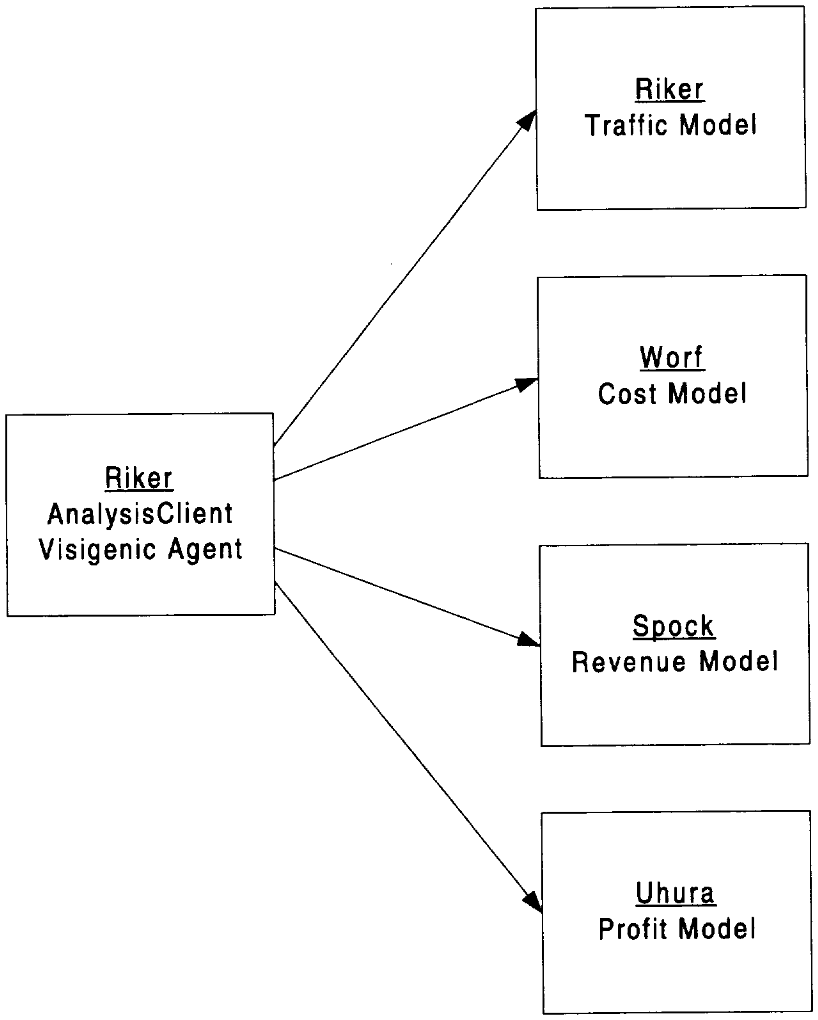
In Figures 5-29 through 5-32, each box represents a server. The underlined name in each box is the name of the server, the boxes on the left are analysis servers and the boxes on the right are model servers. The lines indicate communication paths, and an X through a box indicated that it was shut down during execution.

Single-Analysis Scenario

The single-analysis scenario, depicted in Figure 5-29, demonstrated the following:

- Four models running on four separate machines.
- When the POC analysis was run, each model was called at the appropriate time.
- Two models, cost and revenue, were run simultaneously.

Figure 5-29. Single-Analysis Scenario

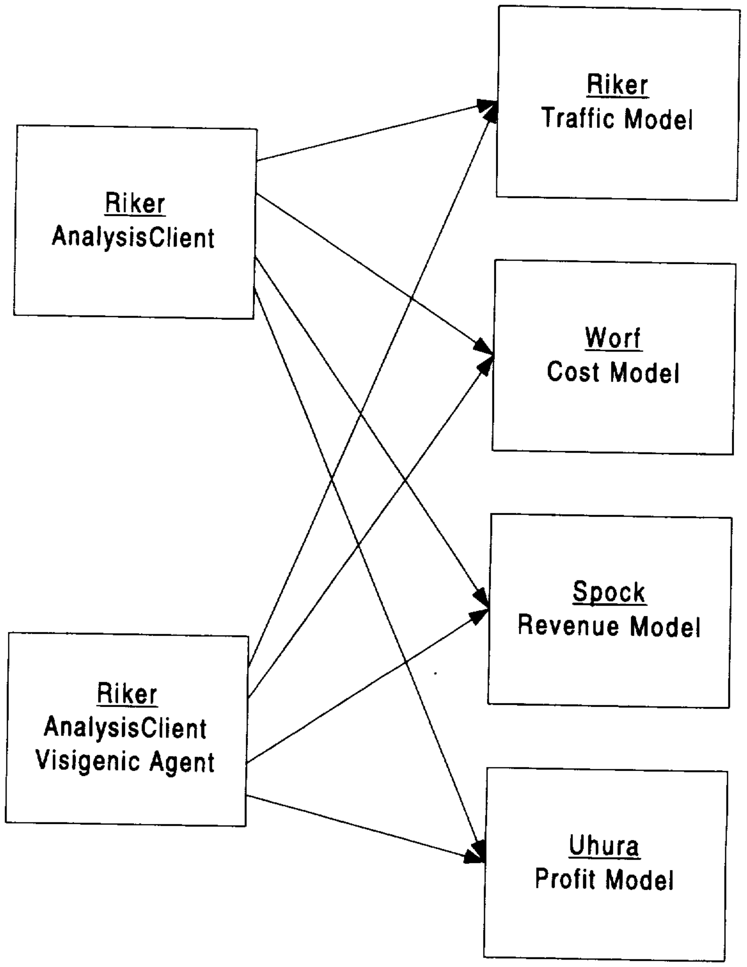


Multiple-Analysis Scenario

The multiple-analysis scenario, depicted in Figure 5-30, demonstrated the following:

- Two instances of the POC analysis ran simultaneously.
- Each analysis called each model server simultaneously.
- Each model server ran two copies of the requested model.

Figure 5-30. Multiple-Analysis Scenario



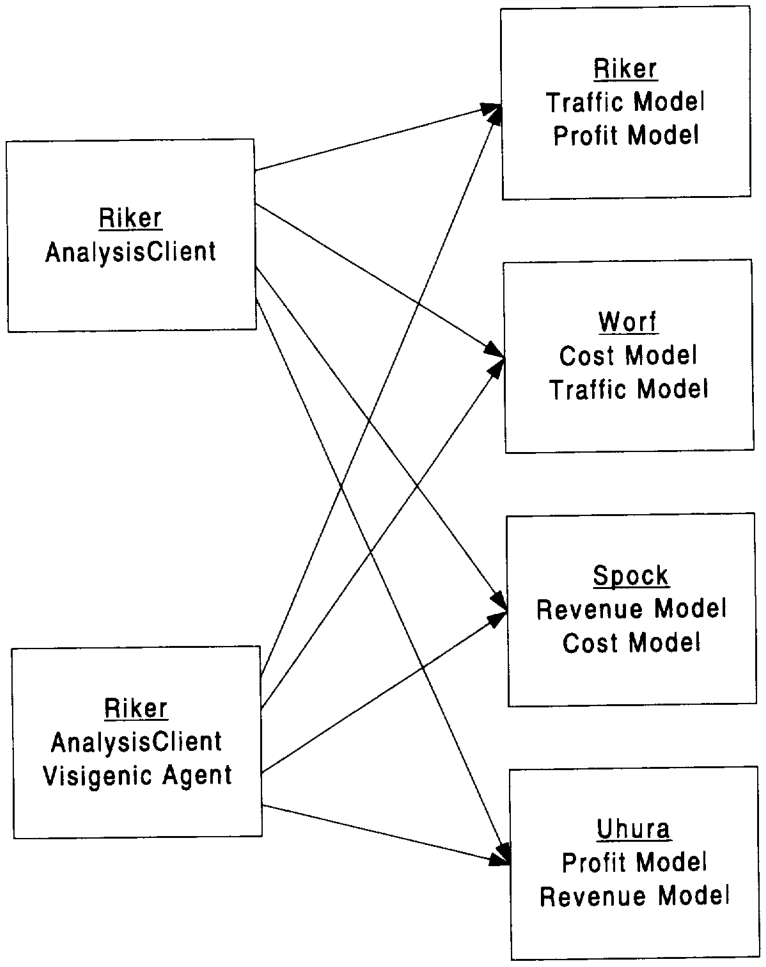
Load-Balancing Scenario

The load-balancing scenario, depicted in Figure 5-31, demonstrated the following:

- Two different models were run on each server.
- Two instances of the POC analysis were run simultaneously.

- Each analysis chose a different instance of each model.

Figure 5-31. Load-Balancing Scenario

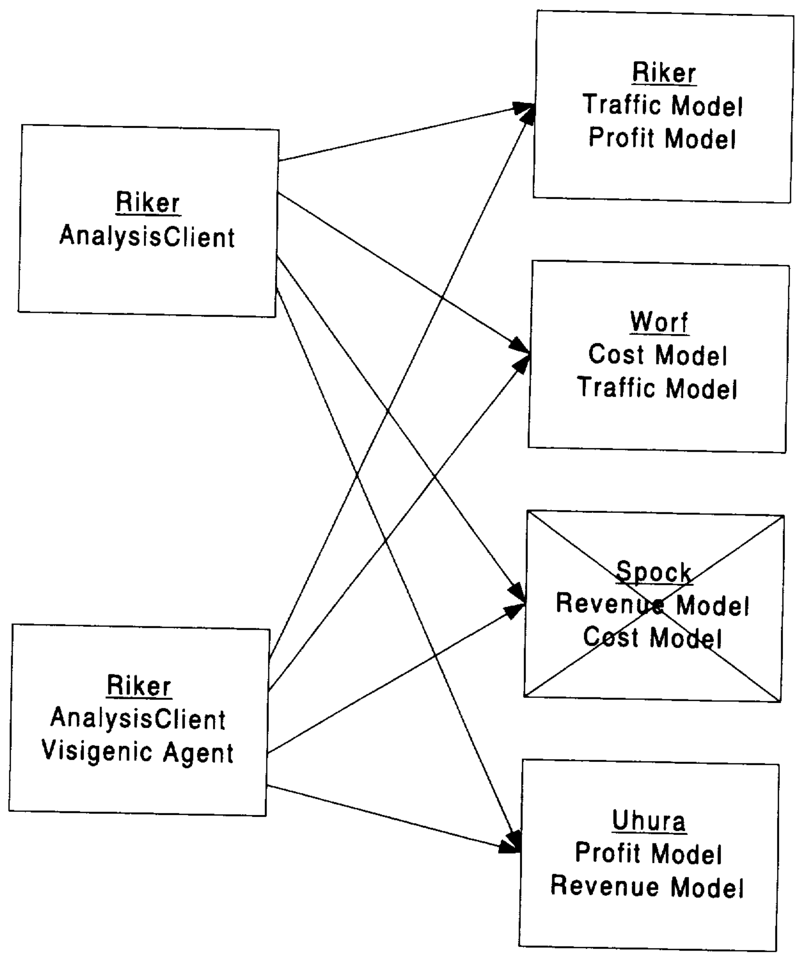


Fault-Tolerance Scenario

The fault-tolerance scenario, depicted in Figure 5-32, demonstrated the following:

- Two instances of the POC analysis ran simultaneously.
- During execution, one of the model servers was shutdown.
- The analyses recovered and continued execution.

Figure 5-32. Fault-Tolerance Scenario



POC User GUI evaluation

In addition to the ASAC EA POC, a mockup Java GUI client was developed as a prototype for the end-user interface to the EA system. The mock-up was developed to gather early user feedback on the general look feel, and navigational metaphors, and for demonstrating to NASA along with the ASAC EA POC.

The GUI created for the ASAC EA POC was a nonfunctional prototype and was a standalone (not connected to the server).

Results of the POC

The ASAC EA POC test procedures were successfully completed on February 23, 1998. The ASAC EA POC and GUI were demonstrated to and accepted by NASA in March 1998. The ASAC EA POC

- Successfully integrated distributed models in various configurations
- Demonstrated validity of system design
- Demonstrated single and multiple analyses
- Demonstrated load balancing and fault tolerance
- Works like final system will work
- Produced reusable products that will be used in the final system.

DSSA Substage 5-4: Review and Iterate

Review and iterate the items developed in DSSA stage 5.

Chapter 6
ASAC EA Beta Version

As mentioned in Chapter 5, this chapter discusses each of the applicable areas of DSSA stages 4 and 5 of the ASAC EA Beta version.

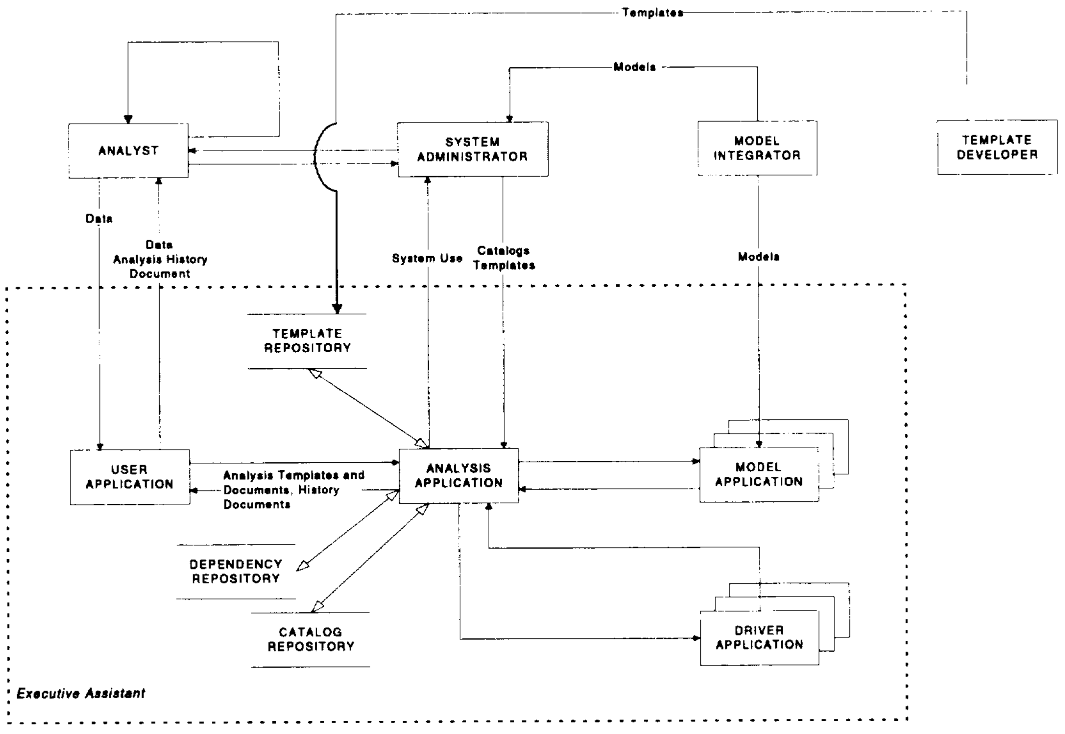
The ASAC EA Beta version expands on the ASAC EA POC. As described in Chapter 5, the Beta version will meet all of the ASAC EA system requirements except optimization and security. Figure 6-1 shows the context diagram of the ASAC EA Beta version.

The analyses that are available in the ASAC Executive Assistant (First Generation), at http://www.asac.lmi.org/eawelcome.html, will be incorporated into the ASAC EA Beta version. The analyses are

- ◆ Aircraft Technology
- ◆ Air Traffic Management.

The ASAC EA Beta version will be demonstrated to NASA in November 1998.

Figure 6-1. Beta Version Context Diagram



Beta Version Goals

The ASAC EA POC was developed to address the high risk areas of the ASAC EA system. The goals for the ASAC EA Beta version were to expand on the ASAC EA POC by

- developing the client (user) application,
- expanding the analysis and model applications,
- integrating the client and analysis applications, and
- creating and fielding a pre-release version of the ASAC EA for initial testing.

Review and Iterate DSSA Substage 2-8: Define Assumptions

The assumptions defined in the design phase of the project and described in the *Aviation System Analysis Capability Executive Assistant Design* still apply. In addition, the following assumptions were made while implementing the Beta version:

- ASAC EA Clients must be capable of running a Java virtual machine, version 1.1.6 or greater, and must be able to connect to the AnalysisServer over the Internet by using CORBA protocols as implemented by Inprise's (previously Borland/Visigenic) VisiBroker ORB.

Review and Iterate DSSA Substage 2-9: Define Issues

Issues remaining from POC are as follows:

- What are the space constraints on user systems (maximum size for the user application)? What is the target size of the analysis application?

  The analysis client is designed to run on a Pentium-class machine with 32 megabytes (MBs) of random access memory (RAM) or greater. Size of the application, including the Java virtual machine, should not exceed 20 MB of disk space.

- Do we use a database or some other mechanism (flat files) for storing analysis and model specifications? If we use a database, is it relational, OO, or a hybrid?

An object-oriented database will be used to store analysis and model specifications as well as other data used by the system.

- When a model fails because of an error, how is its parent analysis notified? (How do we handle errors in a multithreaded environment?)

  Use the Subject/Observer paradigm and observe when the model changes its state to error. Also propagate this change to the client by using a distributed callback mechanism very similar to the Subject/Observer paradigm.

New issues were identified while developing the ASAC EA Beta version. They were:

- How does ASAC handle inputs or outputs that do not match its current two-dimensional format for data elements?

  Data that is greater than two-dimensions, or has a variable number of dimensions, does not fit well into the current data structure defined by the ASAC POC. The structure could be expanded to handle more than two dimensions, but this would make editing and viewing the data difficult to the user. Instead, for the Beta, we added the facility to link data elements together in ways that can mimic n-dimensional data structures. This is a fairly easy addition to implement, and it also keeps the interface simple for the user since they only deal with two dimensions at a time.

- How does ASAC handle binary (i.e., graphics) and other types of data that models may return as output beyond the currently specified types of data?

  This will be investigated further after the beta, but the current thinking is to expand the CORBA interface definition by using unions that could be used to specify additional types of data elements, such as binary data.

DSSA Stage 4—Develop and Refine Beta Version Analysis and Model Application Domain Models

In addition to the POC domain models created for the analysis and model applications, an entire part of the ASAC EA system, the user application, or GUI client is new to the ASAC EA Beta version. The user application has its own set of domain models. For easy comparison and viewing, the complete set of analysis and model application domain models are followed by the complete set of user application domain models.

Analysis and Model Application domain models that were developed for the POC and documented in Chapter 5 were refined during Beta version development. The domain models that were refined are:

- 4-3 Use case diagrams
- 4-4 Interaction diagrams

  Sequence diagrams
- 4-5 Package diagrams
- 4-6 Class diagrams
- 4-7 State diagrams
- 4-8 Deployment diagrams.

New classes were added while implementing the Beta version to support new features, such as breakpoints, and integrate the GUI client and the server. The new classes are:

- Breakpoint
- AnalysisServer
- AnalysisServer_i
- AnalysisObserver
- ConversionUtils.

More detail about these classes will be in the discussion of the DSSA Substages in this chapter.

DSSA Substage 4-3: Develop Use Case Diagrams

No changes were made to the use case diagrams for the ASAC EA Beta version.

DSSA Substage 4-4: Develop Interaction Diagrams

No fundamental changes were made to the relationships between objects, so no changes to the existing interaction diagrams were necessary for the ASAC EA Beta version. One new set of interaction diagrams (performing a DataRelationship) was added to show the interaction between the Breakpoint and DataRelationship classes.

PERFORMING A DATARELATIONSHIP

Figure 6-2. Performing a DataRelationship Sequence Diagram

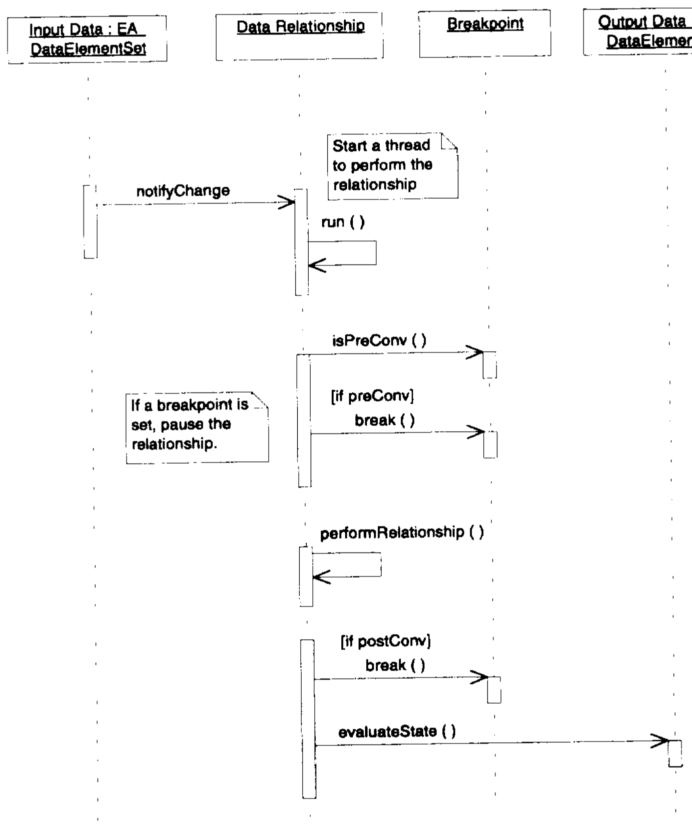
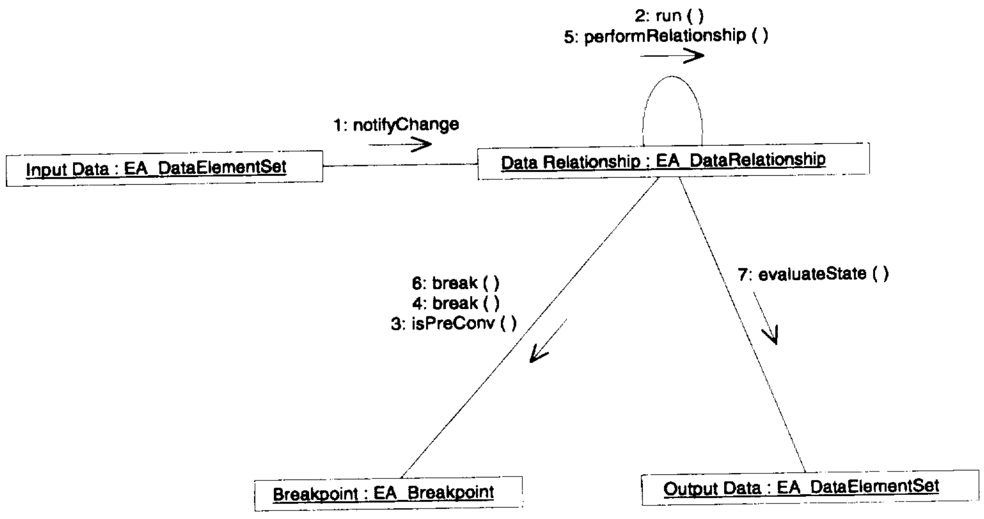

Input Data : EA_DataElementSet
Data Relationship
Breakpoint
Output Data : EA_DataElementSet
Start a thread to perform the relationship
notifyChange
run ( )
isPreConv ( )
If a breakpoint is set, pause the relationship.
[if preConv] break ( )
performRelationship ( )
[if postConv] break ( )
evaluateState ( )

Figure 6-3. Performing a DataRelationship Collaboration Diagram


2: run ( )
5: performRelationship ( )
1: notifyChange
Input Data : EA_DataElementSet
Data Relationship : EA_DataRelationship
6: break ( )
4: break ( )
3: isPreConv ( )
7: evaluateState ( )
Breakpoint : EA_Breakpoint
Output Data : EA_DataElementSet

DSSA Substage 4-5: Develop Package Diagrams

Package diagrams are used for readability purposes only. When a design becomes large, it is convenient to separate groups of classes into separate packages. The Beta server design has been divided into nine class packages:

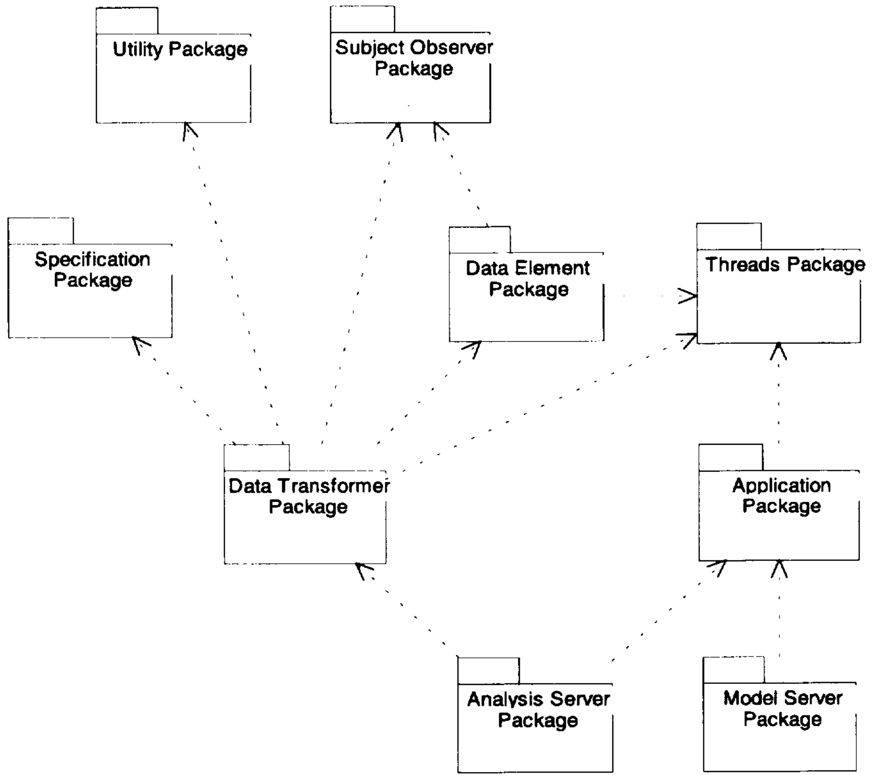
- Subject Observer
- Specification
- Data Transformer
- Data Element
- Threads
- Utility
- Application
- Analysis Server
- Model Server.

Figure 6-4 shows the Beta server package diagram. The diagram is almost identical to the POC package diagram except that the Analysis Client package is now the Analysis Server package. The GUI client has taken the place of the Analysis Client package. The new Analysis Server package handles the creation and execution of analyses, as well as the communication with the GUI client. The dependencies among the classes are denoted by the dashed lines. The dependencies are the following:

- The DataTransformer package depends on the Specification package to read in Analysis and Model specifications.
- The DataTransformer package depends on the DataElement package to hold the inputs and outputs for data transformers.
- The DataTransformer and Data Element packages depend on the Subject Observer package to handle notify DataTransformers, DataRelationships, and DataElement Sets of state changes in other objects that they depend on.
- The DataTransformer and Data Element packages depend on the Threads package to execute models in parallel and to provide synchronization between threads and mutually exclusive access to shared data.

- The DataTransformer package depends on the Utility package to provide miscellaneous utility functions.
- The Application package depends on the Threads package to implement a thread-safe asynchronous-signal-handling thread.
- The Analysis Server and Model Server packages depend on the Application package to handle basic application functions, such as signal handles, error logging, command-line parsing, as well as functions such as initialization and object registration, specific to CORBA clients and servers.
- The Analysis Server package depends on the DataTransformer package to coordinate the execution of analyses consisting of multiple potentially distributed models.

Figure 6-4. Beta Version Package Diagram



DSSA Substage 4-6: Develop Class Diagrams

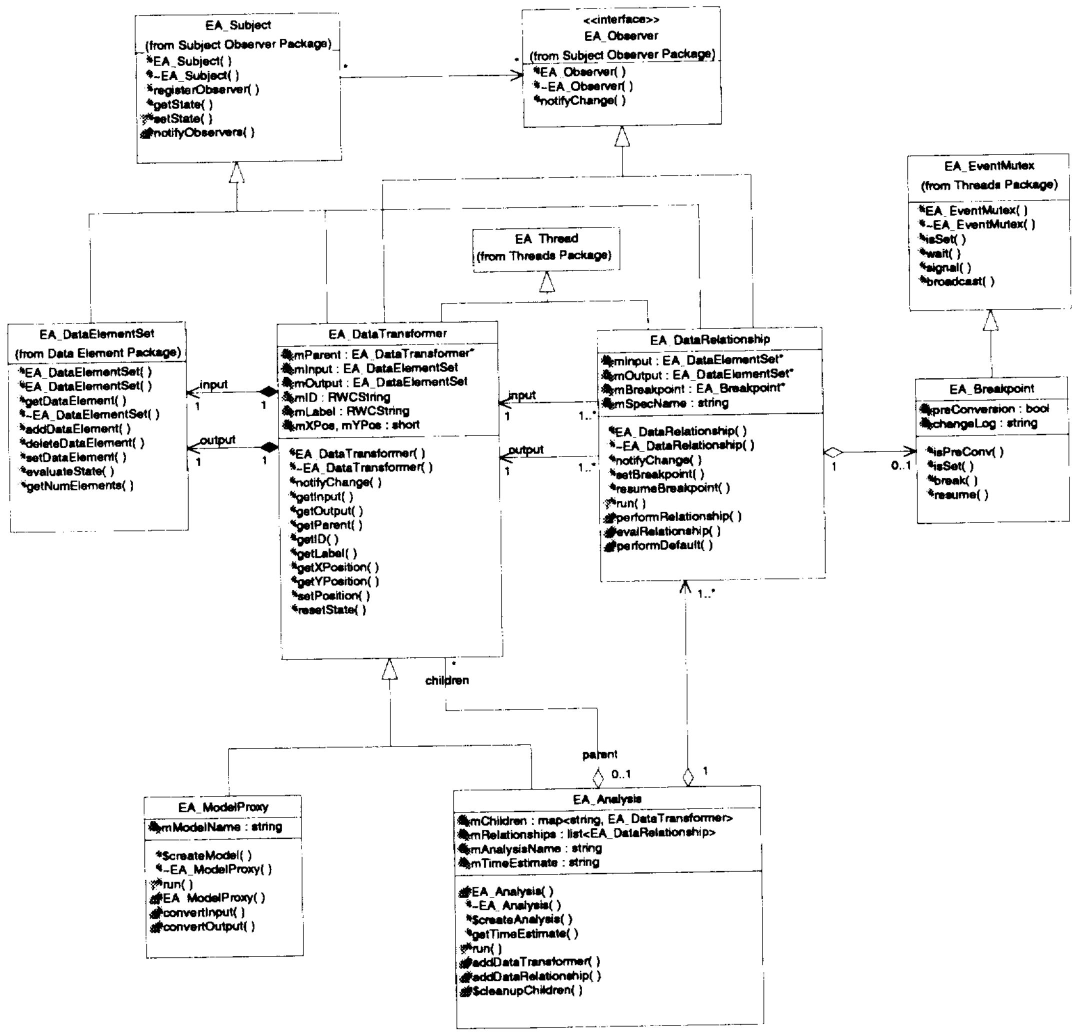
New classes were developed, and existing classes were modified while developing the ASAC Beta to support new functions. The class diagrams for the new or modified classes will be shown in accordance with their package. Classes that have not changed significantly from the POC documented in Chapter 5 will not be documented here.

Data Transformer Package

The DataTransformer package contains the classes that are used for executing an analysis. The Subject, Observer, DataElementSet (DES), and Thread classes are shown on the class diagram to illustrate their relationships with the classes in this package. They are not a part of the DataTransformer package. All DataTransformers inherit from the Subject and Observer classes and contain two DataElementSets; one acts as its input, and the other its output. An Analysis is a special kind of DataTransformer, which contains other DataTransformers (Models & Analyses) and DataRelationships. DataRelationships act as links within an analysis that pass data from one DataTransformer to the next. The class diagram for the DataTransformer package is shown in Figure 6-5.

The major changes in the Data Transformer package from the POC are the additions to the DataTransformer and DataRelationship classes to support breakpoints and the integration of the GUI client and the Breakpoint class to support breakpoints. The three classes are described in this section.

Figure 6-5. DataTransformer Class Diagram



DataTransformer

The DataTransformer is an abstraction for a class that transforms input data values into output data values. It has two DataElementSets that contain the input and output values of the transformer. The DataTransformer "watches" its input DES and automatically performs the transformation when all of its inputs have been set (i.e., its input DES changes to the "ready" state). A list of properties and methods for this class can be found in Table 6-1. The DataTransformer inherits from the Subject and Observer classes, so it also contains the properties and methods shown in Tables 5-1 and 5-2.

Table 6-1. Properties and Methods for DataTransformer Class

Private Properties	
mParent : EA_DataTransformer*	A pointer to the parent of this data transformer or NULL if it has no parent.
mInput : EA_DataElementSet	The set of data elements that acts as the input to the data transformer.
mOutput : EA_DataElementSet	The set of data elements that acts as the output of the data transformer.
mID : RWCString	The ID of the analysis that the client uses to refer to the analysis.
mLabel : RWCString	The label or name of the analysis that should be displayed to the user.
mXPos, mYPos : short	The X & Y coordinates that the analysis should be displayed at in the GUI.
Public Methods	
EA_DataTransformer (spec : EA_TransformerSpec&, parent : EA_DataTransformer*) : EA_DataTransformer	Constructor. Initializes the input and output DESes based on the TransformerSpec. Also registers as an observer to its input DES.
~EA_DataTransformer () :	Destructor.
notifyChange (subj : EA_Subject&) : void	Virtual method from EA_Observer. If the input DES is ready, start a thread and run the data transformer in it.
getInput () : EA_DataElementSet&	Returns a reference to the data transformer's input DES.
getOutput () : EA_DataElementSet&	Returns a reference to the data transformer's output DES.
getParent () : EA_DataTransformer*	Returns a pointer to the transformer's parent analysis, or null if there isn't one.
getID () : RWCString&	Returns the ID that the clients uses to refer to this DataTransformer.
getLabel () : RWCString&	Returns the label displayed to the user for this DataTransformer.
getXPosition () : short	Returns the X position that the DataTransformer should be displayed at.
getYPosition () : short	Returns the X position that the DataTransformer should be displayed at.
setPosition (x : short, y : short) : void	Sets the X & Y position that the DataTransformer should be displayed at.
resetState () : void	Resets the state of the DataTransformer.

DataRelationship

The DataRelationship class acts as a link between two DataTransformers. It waits for the one of the DESs of the input DataTransformer to be set, gets its data values, performs any necessary data transformation or conversion, and sets the values in the one of the DESs of the target DataTransformer. If a breakpoint is set on the DataRelationship, the DataRelationship pauses until the breakpoint is resumed. A list of properties and methods for this class can be found in Table 6-2.

Table 6-2. Properties and Methods for DataRelationship Class

Private Properties	
mInput : EA_DataElementSet*	The DES that acts as the input to the data relationship.
mOutput : EA_DataElementSet*	The DES that the data relationship writes its output to.
MBreakpoint : EA_Breakpoint*	The breakpoint object for this relationship, or null if no breakpoint is set.
MSpecName : string	The name of the data relationship specification (if any) to use for this relationship.
Public Methods	
EA_DataRelationship (in : EA_DataTransformer&, out : EA_DataTransformer&, specName : const RWCString& = "") : EA_DataRelationship	Constructor.
NotifyChange (subj : EA_Subject&) : void	If the input DES is ready, perform the relationship and set the values of the output DES.
~EA_DataRelationship () :	Destructor.
SetBreakpoint (type : EA_BreakType) : void	Sets a breakpoint on the data relationship. The argument specifies whether the breakpoint should apply before or after the relationship is executed.
ResumeBreakpoint () : void	Resumes execution of the data relationship if a breakpoint is set.

Breakpoint

The Breakpoint class provides the ability to pause analyses at specified points during execution. The Breakpoint class inherits from the EventMutex class, which allows the Breakpoint class to suspend execution of the current thread and then resume when an event is received. A list of properties and methods for this class can be found in Table 6-3.

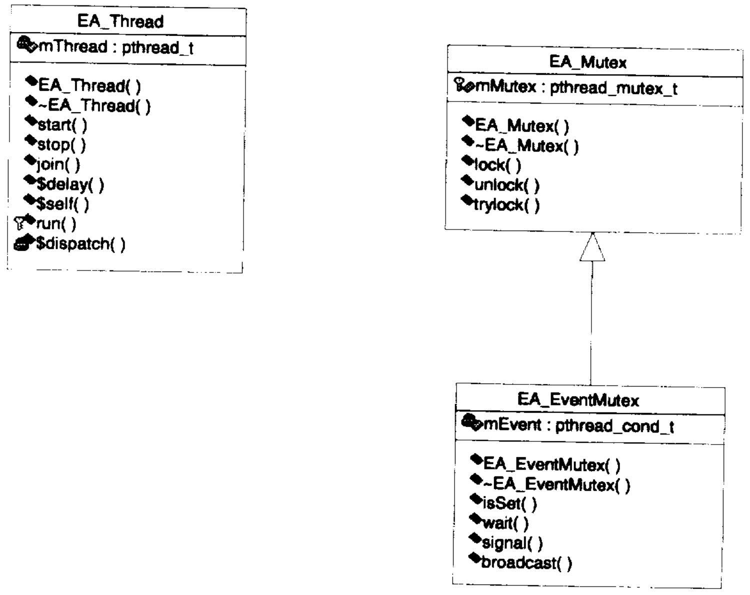
Table 6-3. Properties and Methods for Breakpoint Class

Private Properties	
mPreConversion : bool	Specifies whether the breakpoint should be applied before (if true) or after (if false) the DataRelationship is performed.
Public Methods	
isPreConv () : bool	Returns whether the breakpoint should be set before or after the data relationship (including any necessary conversion) is executed.
isSet () : bool	Returns whether or not the breakpoint is currently "set".
break () : void	Breaks the current thread of control (i.e. the data relationship which called the breakpoint) until resume is called.
resume () : void	Resumes the breakpoint, which allows the data relationship which called it to continue.

THREADS PACKAGE

The Threads package contains classes that support multithreaded execution. They are used primarily to provide the ability to execute models in parallel. The classes act as object-oriented wrappers that support basic multithreading. The Thread class acts as a base class for classes that require a separate thread of execution in a program. The Mutex class provides a locking mechanism for classes that can be used by more than one thread at a time and must ensure that certain operations are executed by only one thread at a time. The EventMutex class is used to block a thread until an external "event" occurs. The only change to this package for the Beta version was the addition of the EventMutex class to support breakpoints. The class diagram for the Threads package is shown in
Figure 6-6.

Figure 6-6. Threads Class Diagram



EventMutex Class

The EventMutex class is used to suspend the execution of one or more threads until an "event" occurs and is fired by another thread. It is primarily used to support breakpoints where a data relationship is paused until the user selects to resume execution. A list of properties and methods for this class can be found in Table 6-4.

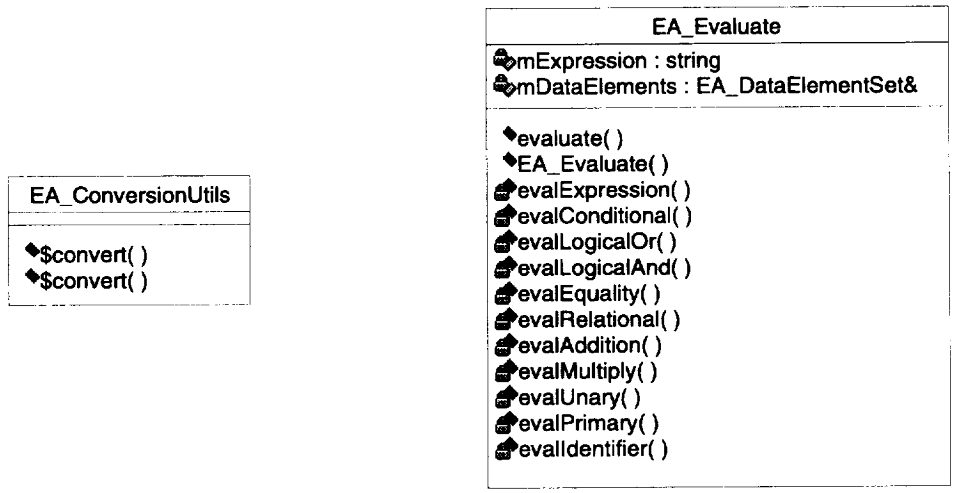
Table 6-4. Properties and Methods for EventMutex Class

Private Properties	
mEvent : pthread_cond_t	The id of the underlying POSIX event which is used to unlock the mutex.
Public Methods	
EA_EventMutex () : EA_EventMutex	Constructor.
~EA_EventMutex ()	Destructor.
isSet () : bool	Returns whether or not the "event" associated with this mutex is true or not.
wait () : void	Suspends execution of the current thread until the mutex is released by another thread calling "signal" or "broadcast."
signal () : void	Triggers the event and allows a single suspended thread to continue execution.
broadcast () : void	Triggers the event and allows all waiting threads to continue execution.

UTILITY PACKAGE

The Utility package contains miscellaneous utility classes used by the Analysis-Server. For the ASAC EA Beta version, a ConversionUtils class was added that is used to convert between CORBA and internal data representations when passing data to and from the GUI client. The package diagram for the Utility package is shown in Figure 6-7.

Figure 6-7. Utility Package Class Diagram



ConversionUtils Class

The ConversionUtils class is used to convert DataElementSets to and from its CORBA counterpart, the DataElement sequences. The ConversionUtils class is used to convert data as they are passed to and from the GUI client. A list of properties and methods for this class can be found in Table 6-5.

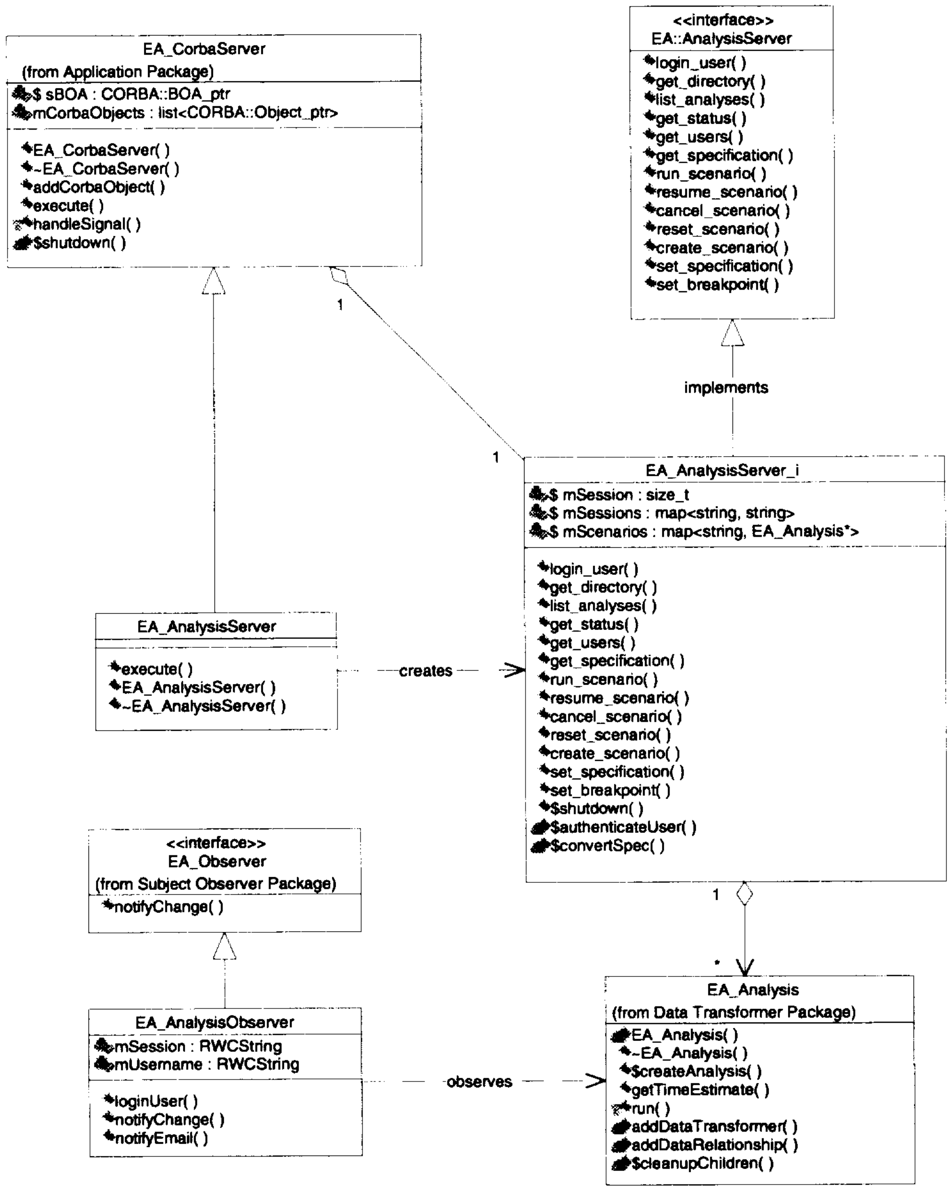
Table 6-5. Properties and Methods for ConversionUtils Class

Public Methods	
convert (in : EA::DataElementSeq&, out : EA_DataElementSet&) : void	Converts a CORBA DataElement sequence into a DataElementSet object.
convert (in : EA_DataElementSet&, out : EA::DataElementSeq&) : void	Converts a DataElementSet object into a CORBA DataElement sequence.

ANALYSIS SERVER PACKAGE

The Analysis Server package acts as the driver for the AnalysisServer component of the ASAC EA. It also acts as the interface to the GUI client component and drives the classes defined in the DataTransformer package that form the backbone of the application. The class diagram for the Analysis Server package is shown in Figure 6-8.

Figure 6-8. Analysis Server Class Diagram



AnalysisServer Class

The AnalysisServer class inherits from the CorbaServer class and provides the driver for the AnalysisServer application. The execute method instantiates an AnalysisServer_i CORBA object, and then waits for requests. A list of properties and methods for this class can be found in Table 6-6.

Table 6-6. *Properties and Methods for AnalysisServer Class*

Public Methods	
execute () : void	Creates an analysis server object and waits for requests.
EA_AnalysisClient () : EA_AnalysisClient	Constructor.
~EA_AnalysisClient () :	Destructor.

AnalysisServer_i Class

The AnalysisServer_i class implements the CORBA AnalysisServer interface and forms the interface to the GUI client. A list of properties and methods for this class can be found in Table 6-7.

Table 6-7. *Properties and Methods for AnalysisServer_i Class*

Private Attributes	
mSession : size_t	Counter used to generate scenario Ids.
mSessions : map<string, string>	List of active scenarios associated with the user that is executing them.
mScenarios : map<string, EA_Analysis*>	List of active scenarios associated with the actual Analysis object responsible for executing the scenario.
Public Methods	
login_user (login : LoginInfo, callback : EA::Client_ptr) : boolean	Logs the user into the system and registers their GUI to receive updates to analysis owned by the given user.
get_directory (login : LoginInfo) : Directory	Returns a recursive listing of all directories and files that the given user has access to.
list_analyses (login : LoginInfo) : Analysis-Seq	Returns a list of active analyses that are "owned" by the given user. The user is authenticated before any data is returned.
get_status (scenario : ObjectID, model ObjectID :) : EA::Status	Returns the current status (RUNNING, DONE, etc.) of the given scenario or model.
get_users (login : LoginInfo) : StringSeq	Returns a list of all known users in the system.
get_specification (login : LoginInfo, scenario : ObjectID, model : ObjectID) : Specification	Returns all the information (the specification) about the given scenario or model that the GUI client needs. This includes lists of inputs and outputs, labels, x & y position, as well as any models/ analyses that it contains and the relationships between them.
run_scenario (scenario : ObjectID) : void	Starts the scenario running if all of required inputs have been set.
resume_scenario (scenario : ObjectID) : void	Resume.
cancel_scenario (scenario : ObjectID) : void	Cancel and close the scenario.
reset_scenario (scenario : ObjectID) : void	Resets the scenario so that it can be executed again.
create_scenario (login : LoginInfo, name : string) : ObjectID	Creates a scenario based on the given analysis. Loads the scenario into memory so that it can be run.

Table 6-7. Properties and Methods for AnalysisServer_i Class (Continued)

set_specification (login : LoginInfo, spec : Specification, scenario : ObjectID, model : ObjectID) : void	Set the inputs, outputs and x & y position on the given analysis or model.
set_breakpoint (scenario : ObjectID, relationship : int, type : EA::BreakType) : void	Set a breakpoint on the given data relationship. The breakpoint type determines the type of breakpoint. "NONE" removes any current breakpoint.
shutdown () : void	Cancel all analysis and shutdown the server.
Private Methods	
authenticateUser (login : EA::LoginInfo) : bool	Returns true if the given username and password refer to a valid user, false otherwise.
convertSpec (dt : EA_DataTransformer, spec : EA::Specification) : void	Extracts the specification of an analysis of model so that it can be passed to the client via CORBA.

AnalysisObserver Class

The AnalysisObserver class notifies the GUI client of state changes in any analyses or models it is running. The AnalysisObserver class also handles e-mail notification if the analysis is running in the background. A list of properties and methods for this class can be found in Table 6-8.

Table 6-8. Properties and Methods for AnalysisObserver Class

Private Attributes	
mSession : RWCString	The session ID of the analysis that this object is observing.
mUsername : RWCString	The username of the user running the analysis. This is used to determine where to deliver updates to the state of the analysis or its children.
Public Methods	
loginUser (username : RWCString, callback : EA::Client_ptr) : void	Registers the client (GUI) callback with the AnalysisObserver object.
notifyChange () : void	Notifies the client of a state change in one of its analyses or models.
notifyEmail () : void	Sends e-mail to the user on completion of an analysis if the user is not logged in.

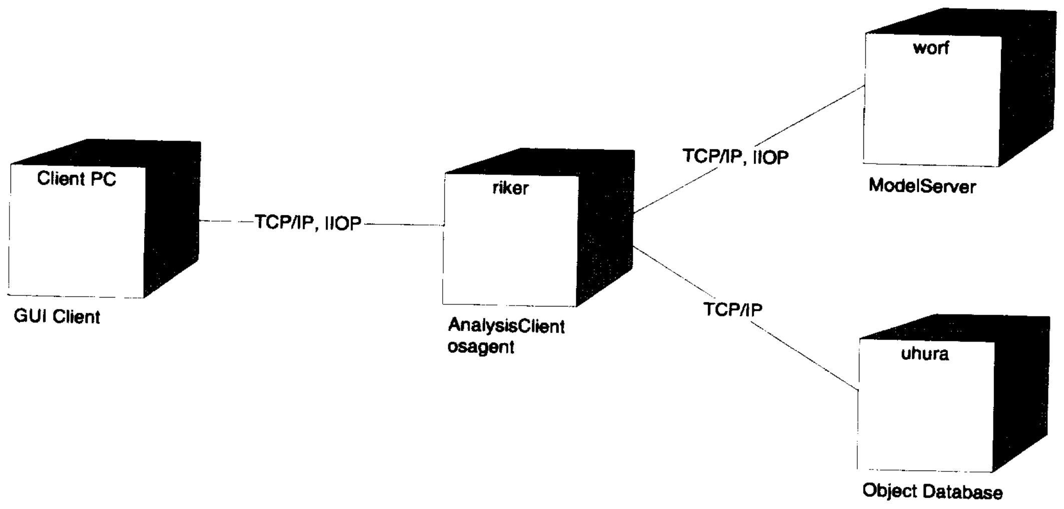
DSSA Substage 4-7: Develop State Diagrams

No changes were made to the state diagrams for the ASAC EA Beta version.

DSSA Substage 4-8: Develop Deployment Diagrams

The Deployment Diagram for the ASAC EA Beta version is shown in Figure 6-9. The model is generic, showing that there will be one or more GUI Clients, an AnalysisServer on riker, the database on uhura, and as a ModelServer on worf that will run multiple Model Applications.

Figure 6-9. Proof of Concept Deployment Diagram



DSSA Stage 4—Develop and Refine Beta Version User Application Domain Models

The following sections contain the user application domain models that were developed for the ASAC EA Beta version client.

Sixteen new classes were developed during implementation of the client part of the Beta version. They are:

- DataElementSet
- DataElement
- LinkSet
- Link
- LinkCanvas
- Orb
- Client

- ModelSet
- Model
- Analysis
- Analysis Manager
- AnalysisNode
- Access
- AnalysisDesktop
- ModelFrame
- MainFrame.

These classes will be discussed in more detail throughout the DSSA Substages in this chapter.

DSSA Substage 4-3: Develop Use Case Diagrams

No changes were made to the use case diagrams for the ASAC EA Beta version.

DSSA Substage 4-4: Develop Interaction Diagrams

The Interaction diagrams developed for the ASAC EA Beta version client are:

- Login
- Loading an Analysis
- Running an Analysis.

LOGIN

Figure 6-10. Login Sequence Diagram

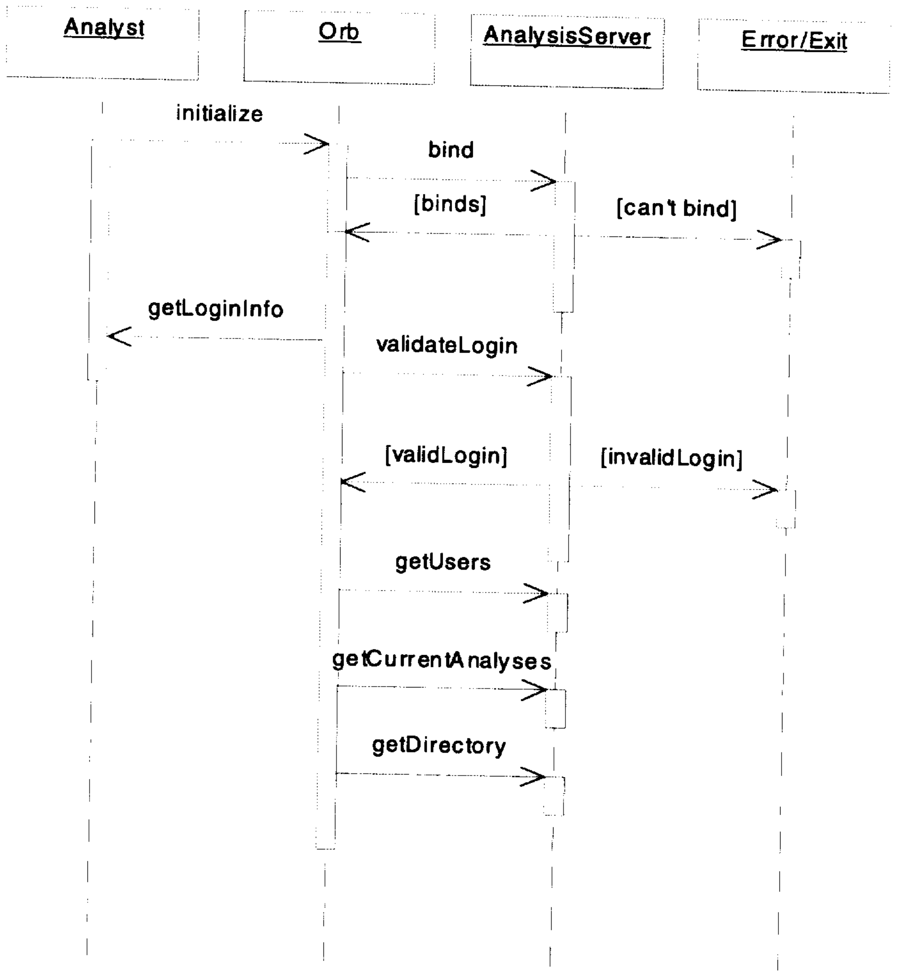
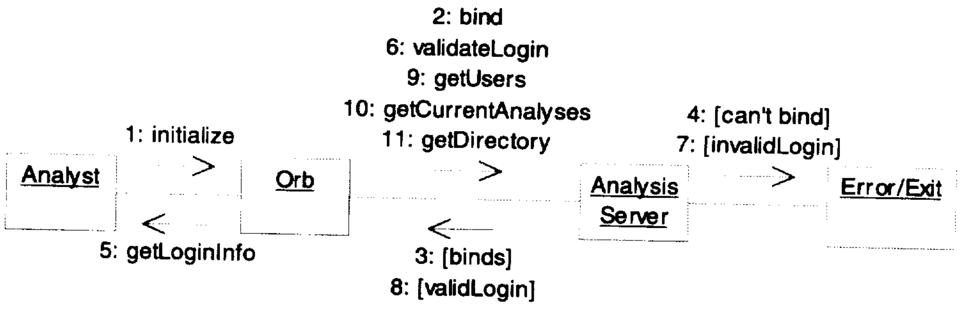


Figure 6-11. Login Collaboration Diagram



LOADING AN ANALYSIS

Figure 6-12. Loading an Analysis Sequence Diagram

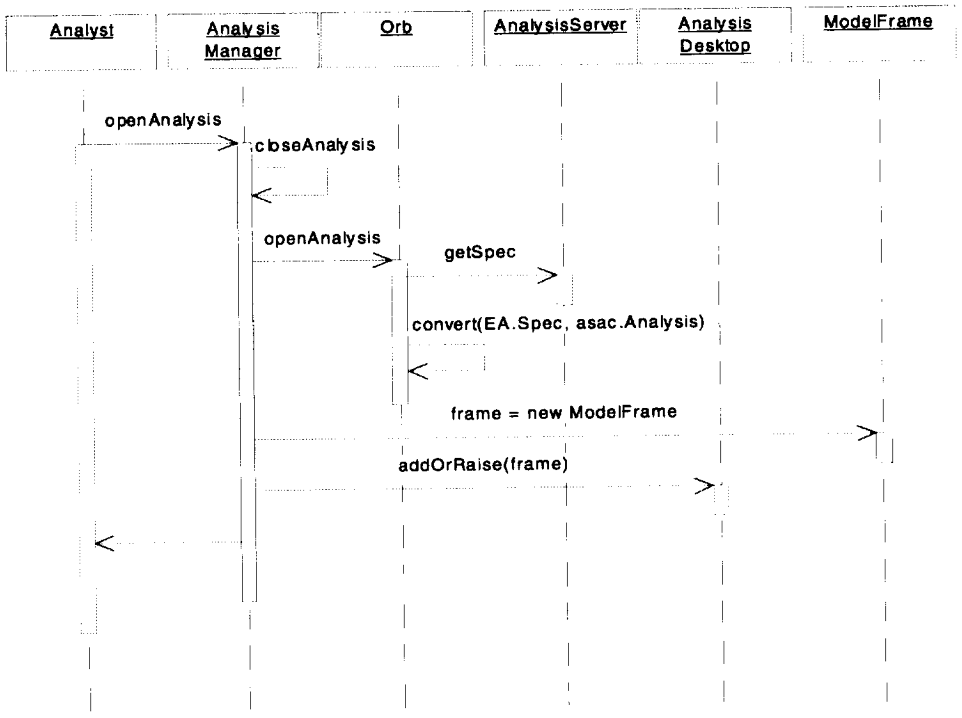
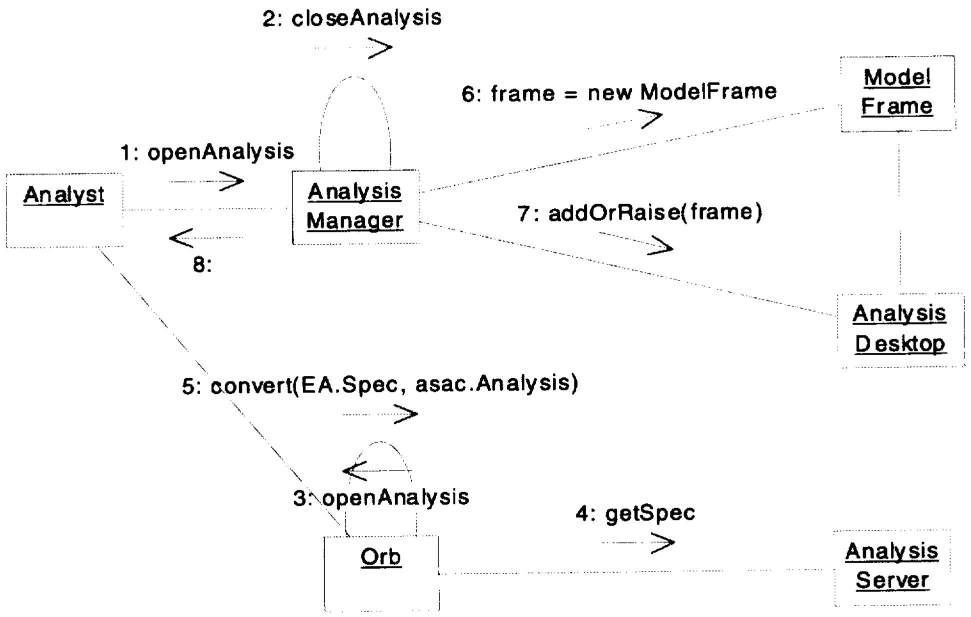


Figure 6-13. Loading an Analysis Collaboration Diagram



RUNNING AN ANALYSIS

Figure 6-14. Running an Analysis Sequence Diagram

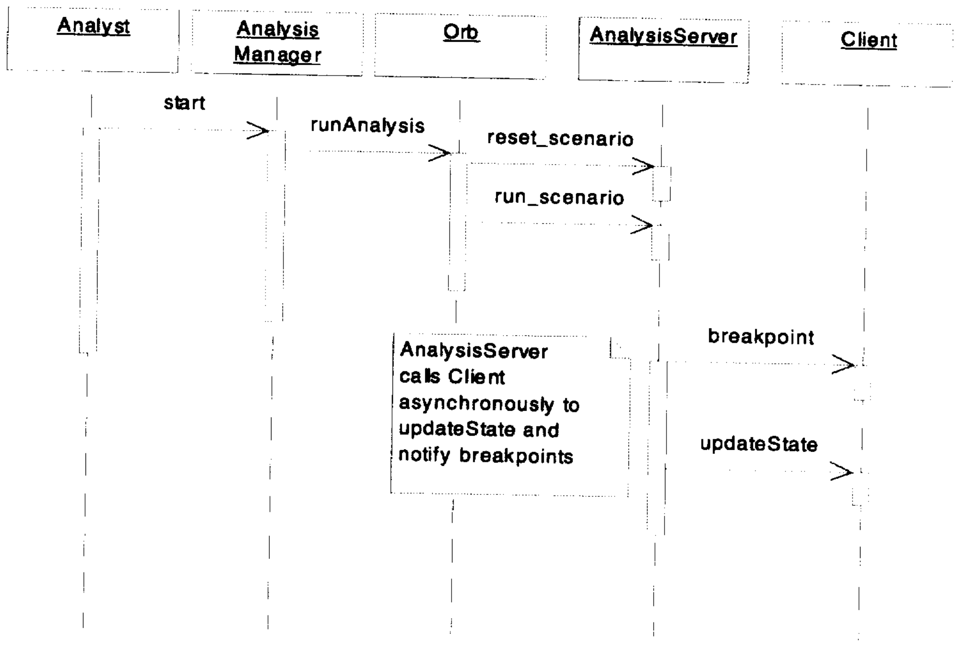
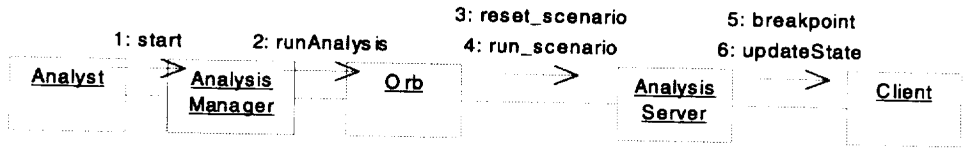


Figure 6-15. Running an Analysis Collaboration Diagram



DSSA Substage 4-5: Develop Package Diagrams

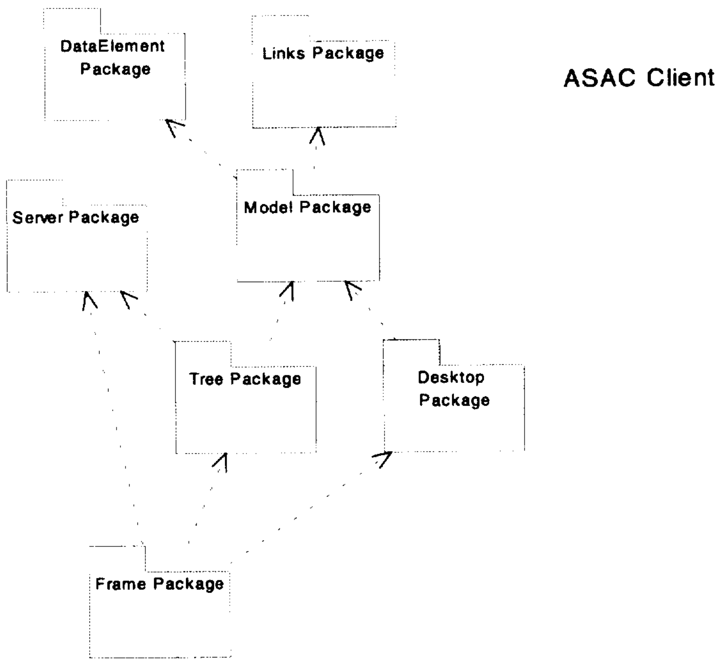
Package diagrams are used for readability purposes only. When a design becomes large, it is convenient to separate groups of classes into separate packages. The client design has been divided into seven class packages:

- The DataElement
- The Links
- The Server
- The Model
- The Tree

- The Desktop
- The Frame

Figure 6-16 shows the POC client package diagram. The package associations among the classes are denoted by the dashed lines.

Figure 6-16. ASAC EA Client Package Diagram



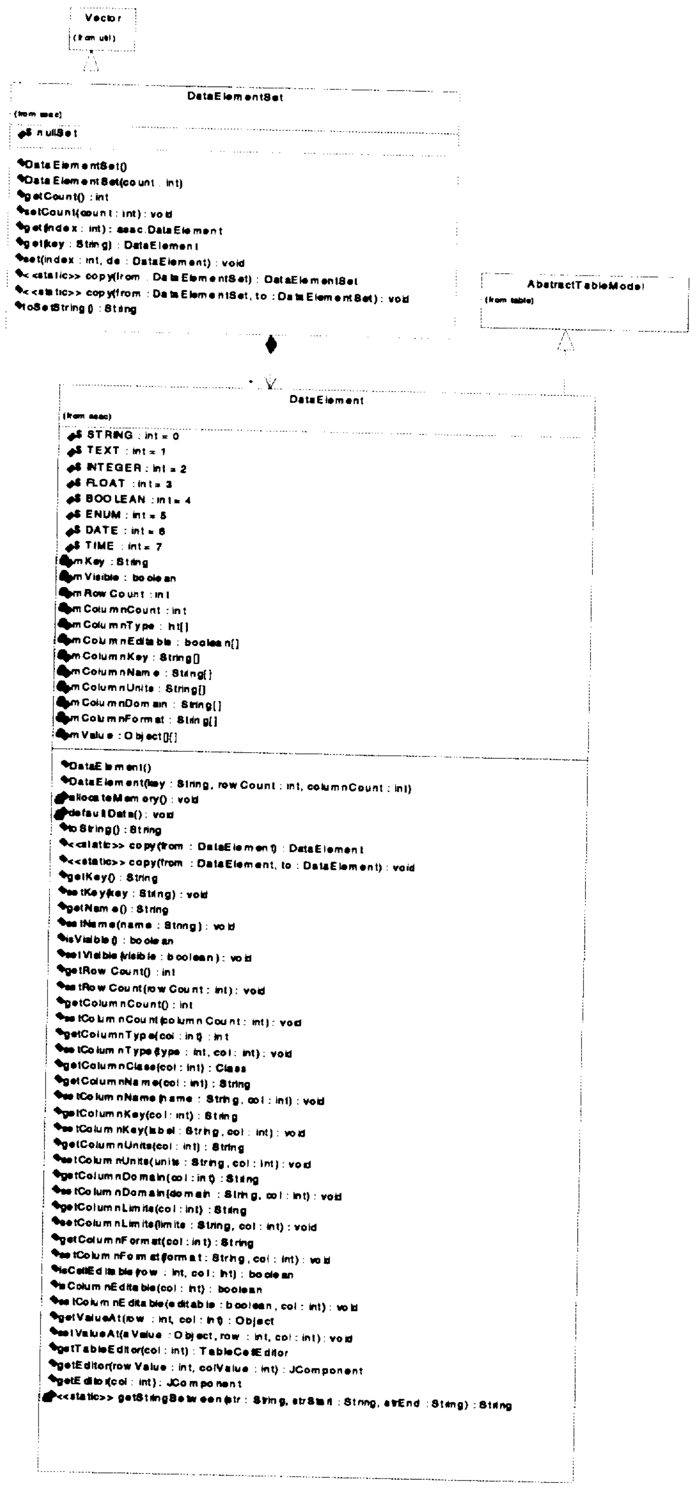
DSSA Substage 4-6: Develop Class Diagrams

Class diagrams are used to illustrate class models and their relationships with other classes. The class diagrams will be shown with their package.

THE DATAELEMENT PACKAGE

The DataElement package contains the classes that provide information about the data elements or parameters for the ASAC models. The package includes the DataElementSet and the DataElement classes. The class diagram is shown in Figure 6-17.

Figure 6-17. DataElement Package Class Diagram


Vector
(from util)
DataElementSet
(from asac)
$ nullSet
DataElementSet()
DataElementSet(count : int)
getCount() : int
setCount(count : int) : void
get(index : int) : asac.DataElement
get(key : String) : DataElement
set(index : int, de : DataElement) : void
<<static>> copy(from : DataElementSet) : DataElementSet
<<static>> copy(from : DataElementSet, to : DataElementSet) : void
toSetString() : String
AbstractTableModel
(from table)
DataElement
(from asac)
$ STRING : int = 0
$ TEXT : int = 1
$ INTEGER : int = 2
$ FLOAT : int = 3
$ BOOLEAN : int = 4
$ ENUM : int = 5
$ DATE : int = 6
$ TIME : int = 7
mKey : String
mVisible : boolean
mRowCount : int
mColumnCount : int
mColumnType : int[]
mColumnEditable : boolean[]
mColumnKey : String[]
mColumnName : String[]
mColumnUnits : String[]
mColumnDomain : String[]
mColumnFormat : String[]
mValue : Object[][]
DataElement()
DataElement(key : String, rowCount : int, columnCount : int)
allocateMemory() : void
defaultData() : void
toString() : String
<<static>> copy(from : DataElement) : DataElement
<<static>> copy(from : DataElement, to : DataElement) : void
getKey() : String
setKey(key : String) : void
getName() : String
setName(name : String) : void
isVisible() : boolean
setVisible(visible : boolean) : void
getRowCount() : int
setRowCount(rowCount : int) : void
getColumnCount() : int
setColumnCount(columnCount : int) : void
getColumnType(col : int) : int
setColumnType(type : int, col : int) : void
getColumnClass(col : int) : Class
getColumnName(col : int) : String
setColumnName(name : String, col : int) : void
getColumnKey(col : int) : String
setColumnKey(label : String, col : int) : void
getColumnUnits(col : int) : String
setColumnUnits(units : String, col : int) : void
getColumnDomain(col : int) : String
setColumnDomain(domain : String, col : int) : void
getColumnLimits(col : int) : String
setColumnLimits(limits : String, col : int) : void
getColumnFormat(col : int) : String
setColumnFormat(format : String, col : int) : void
isCellEditable(row : int, col : int) : boolean
isColumnEditable(col : int) : boolean
setColumnEditable(editable : boolean, col : int) : void
getValueAt(row : int, col : int) : Object
setValueAt(aValue : Object, row : int, col : int) : void
getTableEditor(col : int) : TableCellEditor
getEditor(rowValue : int, colValue : int) : JComponent
getEditor(col : int) : JComponent
<<static>> getStringBetween(str : String, strStart : String, strEnd : String) : String

The DataElementSet Class

A DataElementSet is a collection of DataElements objects keyed by the DataElement's key. The DataElementSet is derived from the dynamic java.util.Vector class with DataElement objects as elements. A list of properties and methods for this class can be found in Table 6-9.

Table 6-9. Properties and Methods for DataElementSet Class

Public Properties:	
nullSet : DataElementSet	An empty set of DataElements.
Public Methods:	
DataElementSet () :	Constructor.
DataElementSet (count : int) :	Constructor with the number of DataElements in the set.
GetCount () : int	Returns the number of DataElements in the set.
setCount (count : int) : void	Sets the number of DataElements in the set.
get (index : int) : DataElement	Returns the DataElement at the specified position.
get(key : String) : DataElement	Returns the DataElement with the specified key.
set (index : int, DataElement : DataElement) : void	Sets the DataElement at the specified position.
copy(from : DataElementSet) : DataElementSet	Creates a copy of the specified DataElementSet.
copy(from : DataElementSet, to : DataElementSet) : void	Copies from the DataElementSet to the specified DataElementSet.
toSetString () : String	Converts the DataElementSet to a string.

The DataElement Class

A DataElement represents a parameter or piece of data to a model. Each DataElement can be a two-dimentional table containing a fixed number of rows and a fixed number of columns. The columns consist of strings, integers, floating point numbers, booleans, enumerated values, dates, or times. A DataElement also can be a text file or a string containing multple lines. DataElements are derived from AbstractTableModel, which provides the basic methods for the TableModel interface. A list of properties and methods for this class can be found in Table 6-10.

Table 6-10. Properties and Methods for DataElement Class

Public Properties:	
STRING : int = 0 TEXT : int = 1 INTEGER : int = 2 FLOAT : int = 3 BOOLEAN : int = 4 ENUM : int = 5 DATE : int = 6 TIME : int = 7	Enumerated (column) types of a DataElement.
Private Properties:	
mKey : String	Unique identifier for this DataElement.
mVisible : boolean	Determines if the DataElement is displayed as one of the model parameters in the GUI.
mRowCount : int	The number of rows in the DataElement.
mColumnCount : int	The number of columns in the DataElement.
mColumnType : int[]	The types of each column (i.e., string and integer).
mColumnEditable : boolean[]	Used by the client/GUI to determine which columns are editable.
mColumnKey : String[]	Unique identifier for each column.
mColumnName : String[]	Used by the client/GUI as the display label for each column.
mColumnUnits : String[]	The value units for each column(i.e., meters and inches).
mColumnDomain : String[]	The domain of each column(i.e., length and time).
mColumnFormat : String[]	A string used to format/validate the value for each column.
mValue : Object[][]	The actual DataElement values for each row and column.
Public Operations:	
DataElement () :	Constructor.
DataElement (key : String, rowCount : int, columnCount : int) :	Constructor with the key, number of rows and columns specified.
toString () : String	Basic method to output the DataElement as a string.
copy (from : DataElement) : DataElement	Create a new copy a DataElement.
copy (from : DataElement, to : DataElement) : void	Copies a DataElement.
getKey () : String setKey (key : String) : void getName () : String setName (name : String) : void	Returns/specifies the desired attribute of the DataElement.

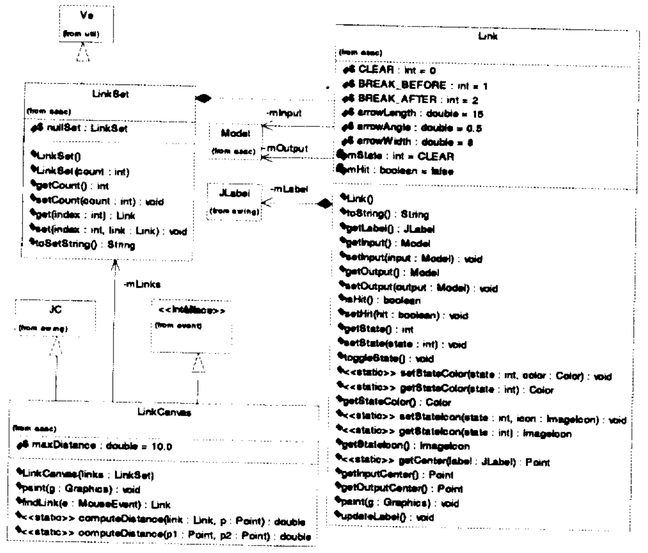
Table 6-10. Properties and Methods for DataElement Class (Continued)

isVisible () : boolean setVisible (visible : boolean) : void getRowCount () : int setRowCount (rowCount : int) : void getColumnCount () : int setColumnCount (columnCount : int) : void getColumnType (col : int) : int setColumnType (type : int, col : int) : void getColumnClass (col : int) : Class getColumnName (col : int) : String setColumnName (name : String, col : int) : void getColumnKey (col : int) : String setColumnKey (label : String, col : int) : void getColumnUnits (col : int) : String setColumnUnits (units : String, col : int) : void getColumnDomain (col : int) : String setColumnDomain (domain : String, col : int) : void getColumnLimits (col : int) : String setColumnLimits (limits : String, col : int) : void getColumnFormat (col : int) : String setColumnFormat (format : String, col : int) : void isCellEditable (row : int, col : int) : boolean isColumnEditable (col : int) : boolean setColumnEditable (editable : boolean, col : int) : void getValueAt (row : int, col : int) : Object setValueAt (aValue : Object, row : int, col : int) : void	
getTableEditor (col : int) : TableCellEditor	Returns a TableEditor for the specified column.
getEditor (rowValue : int, colValue : int) : Jcomponent	Returns an editor for the specified row and column of the DataElement.
getEditor (col : int) : Jcomponent	Returns an editor for any element in the specified column.

THE LINK PACKAGE

The Link package contains classes that link ASAC models or analyses to each other. The package contains the LinkSet, Link, and LinkCanvas classes. The class diagram is shown in Figure 6-18.

Figure 6-18. Link Package Diagram


Ve
(from util)
LinkSet
(from asac)
nullSet : LinkSet
LinkSet()
LinkSet(count : int)
getCount() : int
setCount(count : int) : void
get(index : int) : Link
set(index : int, link : Link) : void
toSetString() : String
-mInput
Model
(from asac)
-mOutput
JLabel
(from swing)
-mLabel
-mLinks
JC
(from swing)
<<Interface>>
(from event)
LinkCanvas
(from asac)
maxDistance : double = 10.0
LinkCanvas(links : LinkSet)
paint(g : Graphics) : void
findLink(e : MouseEvent) : Link
<<static>> computeDistance(link : Link, p : Point) : double
<<static>> computeDistance(p1 : Point, p2 : Point) : double
Link
(from asac)
CLEAR : int = 0
BREAK_BEFORE : int = 1
BREAK_AFTER : int = 2
arrowLength : double = 15
arrowAngle : double = 0.5
arrowWidth : double = 8
mState : int = CLEAR
mHit : boolean = false
Link()
toString() : String
getLabel() : JLabel
getInput() : Model
setInput(input : Model) : void
getOutput() : Model
setOutput(output : Model) : void
isHit() : boolean
setHit(hit : boolean) : void
getState() : int
setState(state : int) : void
toggleState() : void
<<static>> setStateColor(state : int, color : Color) : void
<<static>> getStateColor(state : int) : Color
getStateColor() : Color
<<static>> setStateIcon(state : int, icon : ImageIcon) : void
<<static>> getStateIcon(state : int) : ImageIcon
getStateIcon() : ImageIcon
<<static>> getCenter(label : JLabel) : Point
getInputCenter() : Point
getOutputCenter() : Point
paint(g : Graphics) : void
updateLabel() : void

The LinkSet Class

A LinkSet is a collection of Link objects keyed by index. The LinkSet is derived from the dynamic java.util.Vector class with Link objects as elements. A list of properties and methods for this class can be found in Table 6-11.

Table 6-11. Properties and Methods for LinkSet Class

Public Properties:	
nullSet : LinkSet	An empty set of links.
Public Methods:	
LinkSet () :	Constructor.
LinkSet (count : int) :	Constructor with the number of links in the collection.
getCount () : int	Returns the number of links in the collection.
setCount (count : int) : void	Sets the number of links in the collection.
get (index : int) : Link	Returns the link at the specified index.
set (index : int, link : Link) : void	Sets the link at the specified index.
toSetString () : String	Converts the LinkSet to a string.

The Link Class

A Link connects two Models, an input model to the output model. The Link class is similar to the DataRelation class on the server. Breakpoints may be placed before or after the transformation in the link. The link is represented by a line on the analysis graph pane and an arrow is used to indicate the direction of the data transformation. A list of properties and methods for this class can be found in Table 6-12.

Table 6-12. Properties and Methods for Link Class

Public Properties:	
$CLEAR : int = 0 $BREAK_BEFORE : int = 1 $BREAK_AFTER : int = 2	Enumerated states of a link.
$arrowLength : double = 15	The length of the arrow to paint.
$arrowAngle : double = 0.5	The angle (radians) of the arrow to paint.
$arrowWidth : double = 8	The width of the arrow to paint.
Private Properties:	
mState : int = CLEAR	The link state.
mHit : boolean = false	Maintains if the breakpoint has been hit.
Public Methods:	
Link () :	Constructor.
toString () : String	Converts a Link to a string.
getLabel () : Jlabel	Returns a label or icon to symbolize the state of the link.
getInput () : Model	Returns the Input model.
setInput (input : Model) : void	Specifies the input model.
getOutput () : Model	Returns the output model.
setOutput (output : Model) : void	Specifies the output model.
isHit () : boolean	Checks to see if the breakpoint on the link has been hit.
setHit (hit : boolean) : void	Specifies the hitting of the breakpoint on the link.
getState () : int	Returns the enumerated state of the link.
setState (state : int) : void	Specifies the state of the link.
toggleState () : void	Toggles the enumerated state of the link.
setStateColor (state : int, color : Color) : void	Specifies the color for the enumerated state.
getStateColor (state : int) : Color	Returns the color of the specified state.
getStateColor () : Color	Returns the color of the link.
setStateIcon (state : int, icon : ImageIcon) : void	Specifies the icon for the state.
getStateIcon (state : int) : Im-ageIcon	Returns the icon for the specified state.
getStateIcon () : ImageIcon	Returns the icon for the state of the link.
getCenter (label : JLabel) : Point	Internal helper routine needed when drawing the lines to the center of the input and output models.

Table 6-12. Properties and Methods for Link Class (Continued)

getInputCenter () : Point	Returns the center of the input model.
getOutputCenter () : Point	Returns the center of the output model.
paint (g : Graphics) : void	Routine to paint a link on a graphic object.
updateLabel () : void	Updates the label or icon to reflect the current enumerated status of the link.

The LinkCanvas Class

A LinkCanvas is the background canvas of an Analysis graph model frame. The LinkCanvas is used to draw the link lines and handle mouse events for the links. The LinkCanvas is derived from JComponent and implements the MouseListener interface. A list of properties and methods for this class can be found in Table 6-13.

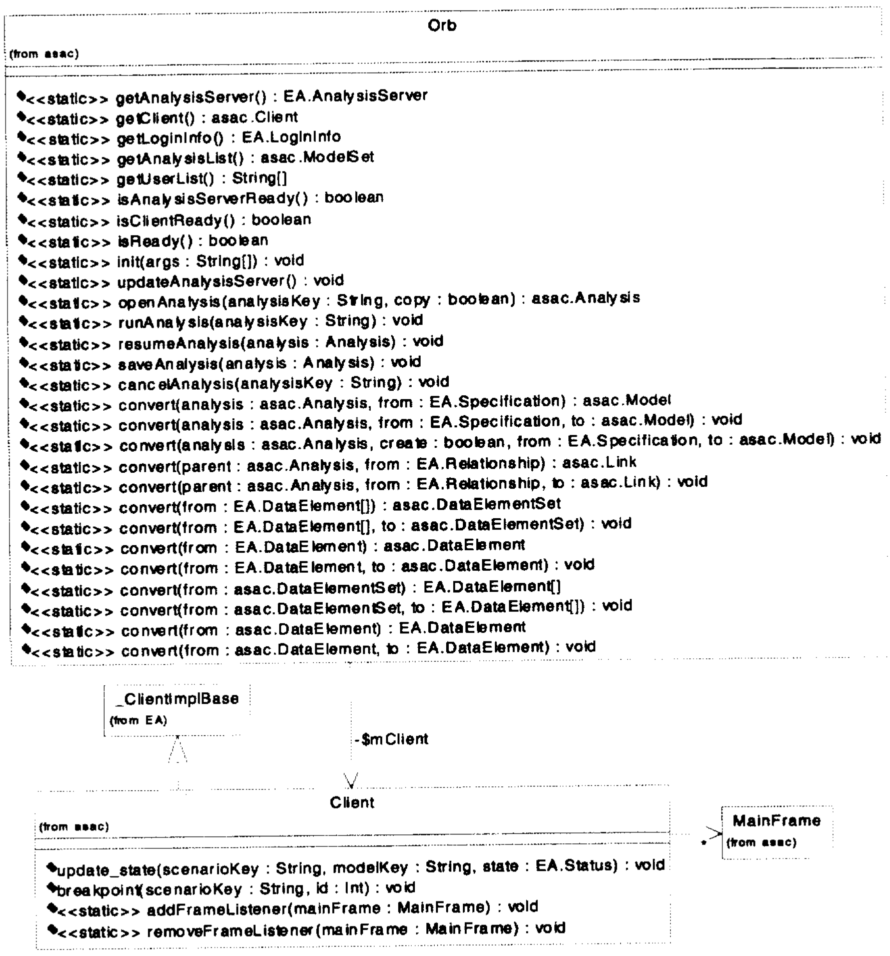
Table 6-13. Properties and Methods for LinkCanvas Class

Public Properties:	
maxDistance : double = 10.0	The maximum pixel distance the mouse can be away from a link.
Public Methods:	
LinkCanvas (links : LinkSet) :	Constructor requires a set of Links.
paint (g : Graphics) : void	Paints the links on the canvas.
findLink (e : MouseEvent) : Link	Routine to determine which link the mouse is near.
computeDistance (link : Link, p : Point) : double	Internal helper routine to compute the distance between a link and a point.
computeDistance (p1 : Point, p2 : Point) : double	Internal routine to compute the distance between to points.

THE SERVER PACKAGE

The Server package contains the classes that deal with the communication between the client and the server. The class diagram is shown Figure 6-19.

Figure 6-19. Server Package Diagram


Orb
(from asac)
<<static>> getAnalysisServer() : EA.AnalysisServer
<<static>> getClient() : asac.Client
<<static>> getLoginInfo() : EA.LoginInfo
<<static>> getAnalysisList() : asac.ModelSet
<<static>> getUserList() : String[]
<<static>> isAnalysisServerReady() : boolean
<<static>> isClientReady() : boolean
<<static>> isReady() : boolean
<<static>> init(args : String[]) : void
<<static>> updateAnalysisServer() : void
<<static>> openAnalysis(analysisKey : String, copy : boolean) : asac.Analysis
<<static>> runAnalysis(analysisKey : String) : void
<<static>> resumeAnalysis(analysis : Analysis) : void
<<static>> saveAnalysis(analysis : Analysis) : void
<<static>> cancelAnalysis(analysisKey : String) : void
<<static>> convert(analysis : asac.Analysis, from : EA.Specification) : asac.Model
<<static>> convert(analysis : asac.Analysis, from : EA.Specification, to : asac.Model) : void
<<static>> convert(analysis : asac.Analysis, create : boolean, from : EA.Specification, to : asac.Model) : void
<<static>> convert(parent : asac.Analysis, from : EA.Relationship) : asac.Link
<<static>> convert(parent : asac.Analysis, from : EA.Relationship, to : asac.Link) : void
<<static>> convert(from : EA.DataElement[]) : asac.DataElementSet
<<static>> convert(from : EA.DataElement[], to : asac.DataElementSet) : void
<<static>> convert(from : EA.DataElement) : asac.DataElement
<<static>> convert(from : EA.DataElement, to : asac.DataElement) : void
<<static>> convert(from : asac.DataElementSet) : EA.DataElement[]
<<static>> convert(from : asac.DataElementSet, to : EA.DataElement[]) : void
<<static>> convert(from : asac.DataElement) : EA.DataElement
<<static>> convert(from : asac.DataElement, to : EA.DataElement) : void
_ClientImplBase
(from EA)
-$mClient
Client
(from asac)
update_state(scenarioKey : String, modelKey : String, state : EA.Status) : void
breakpoint(scenarioKey : String, id : int) : void
<<static>> addFrameListener(mainFrame : MainFrame) : void
<<static>> removeFrameListener(mainFrame : MainFrame) : void
MainFrame
(from asac)

The Orb Class

The Orb is a static class that contains all the operations that a client needs for communicating with the server. The operations include initializing the communication link; retrieving various information from the server; and loading, saving, or executing analyses. A list of properties and methods for this class can be found in Table 6-14.

Table 6-14. Properties and Methods for Orb Class

Public Methods:	
getAnalysisServer () : EA.AnalysisServer	Returns the global instance of the EA.AnalysisServer.
getClient () : asac.Client	Returns the global instance of Client.
getLoginInfo () : EA.LoginInfo	Returns the global instance of EA.LoginInfo.

Table 6-14.. Properties and Methods for Orb Class (Continued)

getAnalysisList () : asac.ModelSet	Returns the global list of analyses currently running.
getUserList () : String[]	Returns a list of users registered to use the ASAC EA.
isAnalysisServerReady () : boolean	Checks to see if the analysis server is ready.
isClientReady () : boolean	Checks to see if the client is ready.
isReady () : boolean	Checks to see if the Orb is ready which implies that the analysis server and the client are both ready.
init (args : String[]) : void	Initializes the Orb.
updateAnalysisServer () : void	Updates the analysis server.
openAnalysis (analysisKey : String, copy : boolean) : asac.Analysis	Opens an analysis on the server.
runAnalysis (analysisKey : String) : void	Executes an analysis on the server.
resumeAnalysis (analysis : Analysis) : void	Resumes an analysis on the server.
saveAnalysis (analysis : Analysis) : void	Saves an analysis on the server.
cancelAnalysis (analysisKey : String) : void	Cancels an analysis on the server.
convert (analysis : asac.Analysis, from : EA.Specification) : asac.Model convert (analysis : asac.Analysis, from : EA.Specification, to : asac.Model) : void convert (analysis : asac.Analysis, create : boolean, from : EA.Specification, to : asac.Model) : void convert (parent : asac.Analysis, from : EA.Relationship) : asac.Link convert (parent : asac.Analysis, from : EA.Relationship, to : asac.Link) : void convert (from : EA.DataElement[]) : asac.DataElementSet convert (from : EA.DataElement[], to : asac.DataElementSet) : void convert (from : EA.DataElement) : asac.DataElement convert (from : EA.DataElement, to : asac.DataElement) : void convert (from : asac.DataElementSet) : EA.DataElement[] convert (from : asac.DataElementSet, to : EA.DataElement[]) : void convert (from : asac.DataElement) : EA.DataElement convert (from : asac.DataElement, to : EA.DataElement) : void	Data converting routines between the client and the server.

The Client Class

The Client is derived from EA._ClientImplBase and contains all the operations that the AnalysisServer needs for communicating with a client. The operations include updating model states or setting breakpoints. All client MainFrames must register and unregister with the global instance of Client to receive updates from the server. A list of properties and methods for this class can be found in Table 6-15.

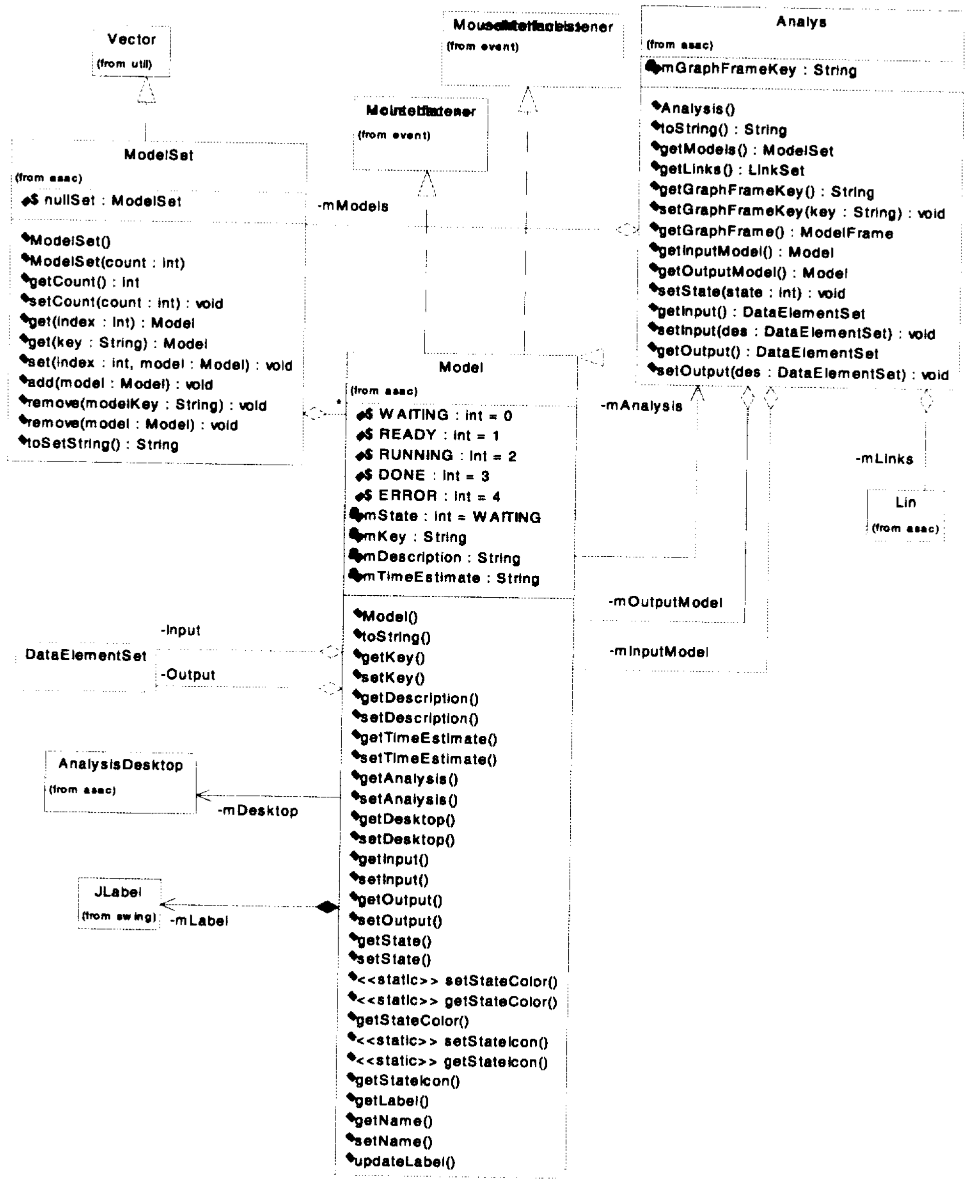
Table 6-15. Properties and Methods for Client Class

Public Methods:	
update_state (scenarioKey : String, modelKey : String, state : EA.Status) : void	Called by the server to update the state of the model specifications.
breakpoint (scenarioKey : String, id : int) : void	Called by the server when a breakpoint is hit.
addFrameListener (mainFrame : MainFrame) : void	Method used to register a mainframe with the client.
removeFrameListener (mainFrame : MainFrame) : void	Method used to unregister a mainframe.

THE MODEL PACKAGE

The Model package contains classes that concern ASAC models and analyses. The package consists of the ModelSet, Model, and Analysis. The class diagram is shown in Figure 6-20.

Figure 6-20. Model Package Diagram



The ModelSet Class

A ModelSet is a collection of Model objects, each with an associated key. The ModelSet is derived from the dynamic java.util.Vector class with Model objects as elements. A list of properties and methods for this class can be found in Table 6-16.

Table 6-16. Properties and Methods for ModelSet Class

Public Properties:	
nullSet : ModelSet	An empty ModelSet.
Public Methods:	
ModelSet () :	Constructor.
ModelSet (count : int) :	Constructor specifying the number of models in the collection.
getCount () : int	Returns the number of model in the collection.
setCount (count : int) : void	Sets the number of modes in the collection.
get (index : int) : Model	Returns the model at the specified index.
get (key : String) : Model	Returns the model with the specified key.
set (index : int, model : Model) : void	Set the model at the specified index.
add (model : Model) : void	Adds a model to the collection.
remove (modelKey : String) : void	Removes the model with the specified key from the collection.
remove (model : Model) : void	Removes the specified model from the collection.
toSetString () : String	Converts the collection to a string.

The Model Class

A Model transforms an input DataElementSet to an output DataElementSet. A model is very similar to the DataTransformer class on the server. The Model class implements the MouseListener and the MouseMotionListener interfaces. A Model also is part of an Analysis and contains a label on the Analysis graph pane. A list of properties and methods for this class can be found in Table 6-17.

Table 6-17. Properties and Methods for Model Class

Public Properties:	
WAITING : int = 0 READY : int = 1 RUNNING : int = 2 DONE : int = 3 ERROR : int = 4	Enumerated states of a Model.
Private Properties:	
mState : int = WAITING	The enumerated state (Waiting, Ready, Running, Done, Error) of the model.
mKey : String	The unique key for the model.

Table 6-17. Properties and Methods for Model Class (Continued)

mDescription : String	The narrative description for the model.
mTimeEstimate : String	The estimated time for the model to execute.
Public Methods:	
Model () :	Constructor.
toString () : String	Converts the model to a string.
getKey () : String	Returns the unique identifier for the model.
setKey (key : String) : void	Specifies the unique identifier for the model.
getDescription () : String	Returns a narrative description for the model.
setDescription (description : String) : void	Sets the narrative description for the model.
getTimeEstimate () : String	Returns the estimated time it takes for the model to run.
setTimeEstimate (timeEstimate : String) : void	Sets the estimated time for the model to run.
getAnalysis () : Analysis	Returns the parent analysis for the model.
setAnalysis (analysis : Analysis) : void	Sets the parent analysis for the model.
getDesktop () : AnalysisDesktop	Returns the desktop for the model.
setDesktop (desktop : AnalysisDesktop) : void	Specifies the desktop for the model.
getInput () : DataElementSet	Returns the input data element set for the model.
setInput (des : DataElementSet) : void	Sets the input data element set for the model.
getOutput () : DataElementSet	Returns the output data element set for the model.
setOutput (des : DataElementSet) : void	Sets the output data element set for the model.
getState () : int	Returns the enumerated state of the model.
setState (state : int) : void	Sets the model state.
setStateColor (state : int, color : Color) : void	Specifies the color for the enumerated state.
getStateColor (state : int) : Color	Returns the color for the specified state.
getStateColor () : Color	Returns the color for the state of the model.
setStateIcon (state : int, icon : ImageIcon) : void	Specifies the icon for the state.
getStateIcon (state : int) : ImageIcon	Returns the icon for the specified state.
getStateIcon () : ImageIcon	Returns the icon for the state of the model.
getLabel () : Jlabel	Returns the label representing the model.
getName () : String	Returns the text on the label that names the model.
setName (name : String) : void	Specifies the text on the label for the model.
updateLabel () : void	Updates the label according to the state of the model.

The Analysis Class

An Analysis is a Model with a set of models and a set of links. An Analysis contains a GraphFrame, a ModelFrame with a graph of the model calling sequence. The Analysis also contains an input and output Model on the Analysis GraphFrame. A list of properties and methods for this class can be found in Table 6-18.

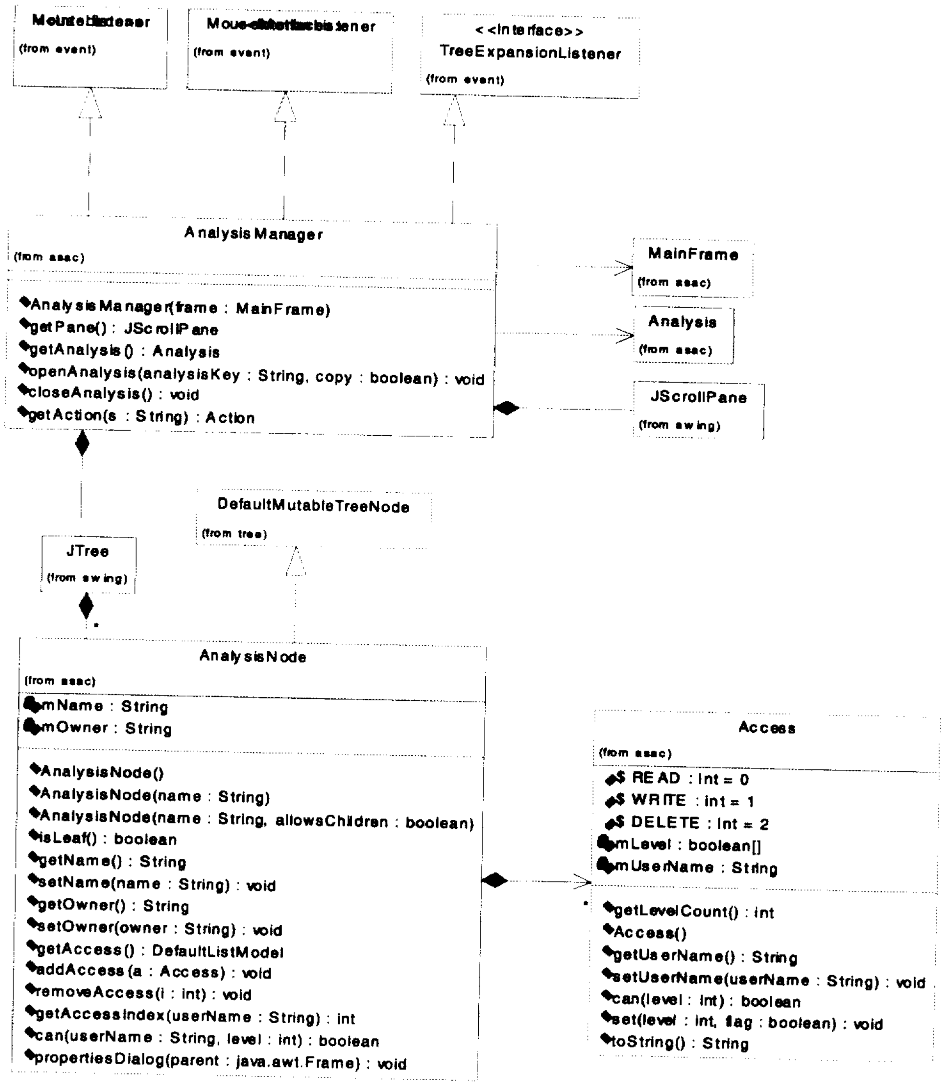
Table 6-18. Properties and Methods for Analysis Class

Private Properties:	
mGraphFrameKey : String	The key for the graph model frame.
Public Methods:	
Analysis () :	Constructor.
toString () : String	Converts an analysis to a string.
getModels () : ModelSet	Returns the models of the analysis.
getLinks () : LinkSet	Returns the links of the analysis.
getGraphFrameKey () : String	Returns the unique key for the analysis graph on the desktop.
setGraphFrameKey (key : String) : void	Sets the unique key for the analysis graph frame on the desktop.
getGraphFrame () : ModelFrame	Returns the ModelFrame with the analysis graph.
getInputModel () : Model getOutputModel () : Model setState (state : int) : void getInput () : DataElementSet setInput (des : DataElementSet) : void getOutput () : DataElementSet setOutput (des : DataElementSet) : void	Override specific model methods.

The Tree Package

The Tree package contains classes that organize the ASAC analyses. The package contains the AnalysisManager, AnalysisNode, and Access. The class diagram is shown in Figure 6-21.

Figure 6-21. Tree Package Diagram


(from event)
(from event)
<<Interface>>
TreeExpansionListener
(from event)
AnalysisManager
(from asac)
◆AnalysisManager(frame : MainFrame)
◆getPane() : JScrollPane
◆getAnalysis() : Analysis
◆openAnalysis(analysisKey : String, copy : boolean) : void
◆closeAnalysis() : void
◆getAction(s : String) : Action
MainFrame
(from asac)
Analysis
(from asac)
JScrollPane
(from swing)
DefaultMutableTreeNode
(from tree)
JTree
(from swing)
AnalysisNode
(from asac)
mName : String
mOwner : String
◆AnalysisNode()
◆AnalysisNode(name : String)
◆AnalysisNode(name : String, allowsChildren : boolean)
◆isLeaf() : boolean
◆getName() : String
◆setName(name : String) : void
◆getOwner() : String
◆setOwner(owner : String) : void
◆getAccess() : DefaultListModel
◆addAccess(a : Access) : void
◆removeAccess(i : int) : void
◆getAccessIndex(userName : String) : int
◆can(userName : String, level : int) : boolean
◆propertiesDialog(parent : java.awt.Frame) : void
Access
(from asac)
$ READ : int = 0
$ WRITE : int = 1
$ DELETE : int = 2
mLevel : boolean[]
mUserName : String
◆getLevelCount() : int
◆Access()
◆getUserName() : String
◆setUserName(userName : String) : void
◆can(level : int) : boolean
◆set(level : int, flag : boolean) : void
◆toString() : String

The AnalysisManager Class

The AnalysisManager class uses the tree paradigm to manage the set of analyses on the server. The AnalysisManager contains a JTree, which organizes the AnalysisTreeNodes. The AnalysisManager implements the MouseListener, the MouseMotionListener, and the TreeExpansionListener and provides a front end for analysis operations like copying, renaming, and displaying properties. A list of properties and methods for this class can be found in Table 6-19.

Table 6-19. Properties and Methods for AnalysisManager Class

Public Methods:	
AnalysisManager (frame : MainFrame) :	Constructor.
getPane () : JScrollPane	Returns the pane of the tree.
getAnalysis () : Analysis	Returns the current analysis.
openAnalysis (analysisKey : String, copy : boolean) : void	Opens the specified analysis.
closeAnalysis () : void	Closes the current analysis.
getAction (s : String) : Action	Returns the specified action.

The AnalysisNode Class

An AnalysisNode is a tree node that contains the information necessary for loading and setting properties of the analysis or scenario on the server. The AnalysisNode is derived from a DefaultMutableTreeNode and provides the name and owner along with permission levels for the scenario or analysis. A list of properties and methods for this class can be found in Table 6-20.

Table 6-20. Properties and Methods for AnalysisNode Class

Private Properties:	
mName : String	The name of the analysis.
mOwner : String	The owner of the analysis.
Public Methods:	
AnalysisNode () :	Constructor.
AnalysisNode (name : String) :	Constructor.
AnalysisNode (name : String, allowsChildren : boolean) :	Constructor.
isLeaf () : boolean	Overrides the isLeaf() method from DefaultMutableTree-Node.
getName () : String	Returns the name of the analysis.
setName (name : String) : void	Specifies the name of the analysis.
getOwner () : String	Returns the owner of the analysis.
setOwner (owner : String) : void	Specifies the owner of the analysis.
getAccess () : DefaultListModel	Returns the access list.
addAccess (a : Access) : void	Adds an access to the list.
removeAccess (I : int) : void	Removes an access from the list.

Table 6-20. Properties and Methods for AnalysisNode Class (Continued)

getAccessIndex (userName : String) : int	Returns the access level of the specified user.
can (userName : String, level : int) : boolean	Test if the user has the access level.
propertiesDialog (parent : java.awt.Frame) : void	Displays and allows the user to edit the properties of the analysis.

The Access Class

The Access class controls the permission levels a user has on a particular analysis or scenario. A list of properties and methods for this class can be found in Table 6-21.

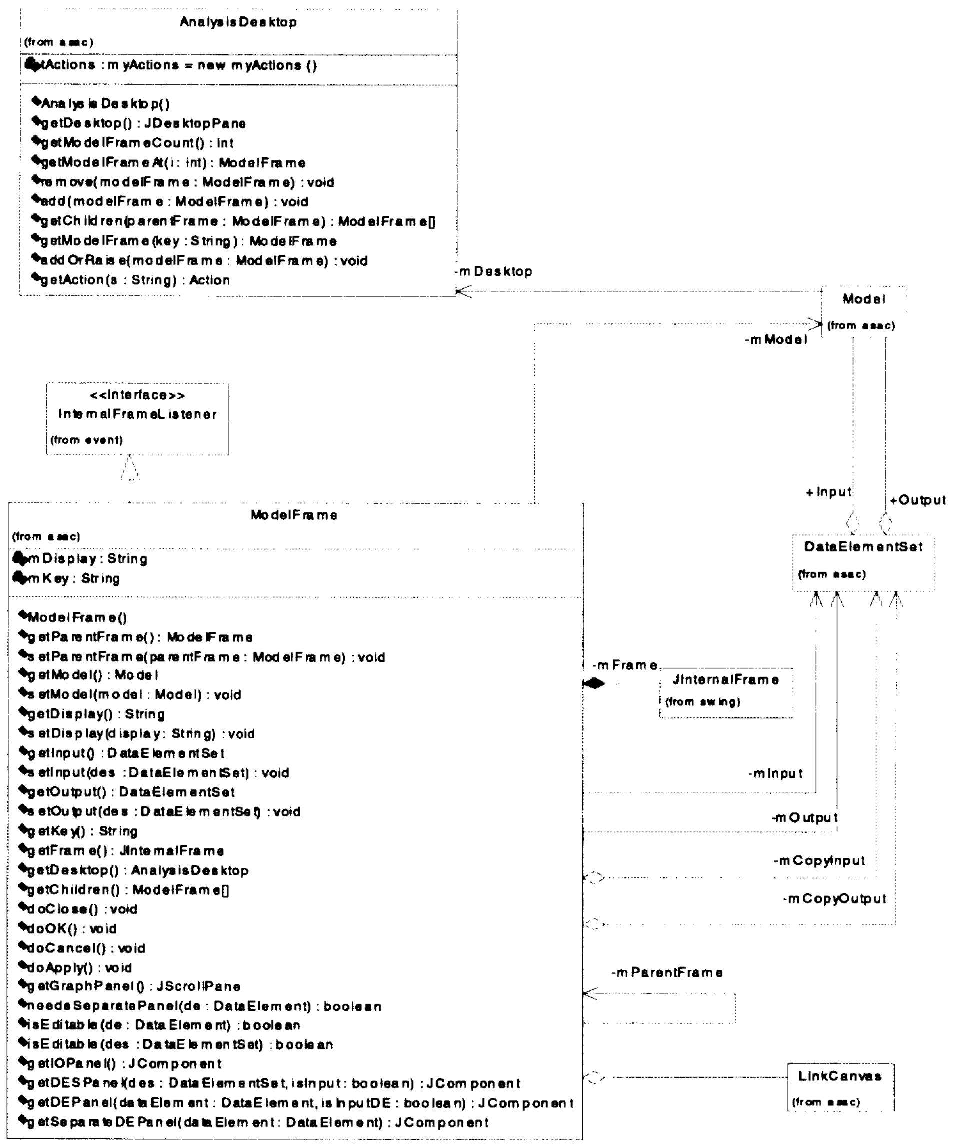
Table 6-21. Properties and Methods for Access Class

Public Properties:	
READ : int = 0 WRITE : int = 1 DELETE : int = 2	Enumerated permission levels.
Private Properties:	
mLevel : boolean[]	The list of permission levels (read, write, and delete).
mUserName : String	The user's name.
Public Methods:	
Access () :	Constructor.
getLevelCount () : int	The number of levels.
getUserName () : String	Returns the user's name.
setUserName (userName : String) : void	Sets the user's name.
can (level : int) : boolean	Tests for user permission on the specified access level.
set (level : int, flag : boolean) : void	Sets or clears the specified access level.
toString () : String	Converts the access to a string.

THE DESKTOP PACKAGE

The Desktop Package contains the AnalysisDesktop and the ModelFrames to be placed on the desktop. The class diagram is shown in Figure 6-22.

Figure 6-22. Desktop Package Diagram


AnalysisDesktop
(from asac)
tActions : myActions = new myActions ()
AnalysisDesktop()
getDesktop() : JDesktopPane
getModelFrameCount() : int
getModelFrameAt(i : int) : ModelFrame
remove(modelFrame : ModelFrame) : void
add(modelFrame : ModelFrame) : void
getChildren(parentFrame : ModelFrame) : ModelFrame[]
getModelFrame(key : String) : ModelFrame
addOrRaise(modelFrame : ModelFrame) : void
getAction(s : String) : Action
-mDesktop
Model
(from asac)
-mModel
<<Interface>>
InternalFrameListener
(from event)
+Input
+Output
DataElementSet
(from asac)
ModelFrame
(from asac)
mDisplay : String
mKey : String
ModelFrame()
getParentFrame() : ModelFrame
setParentFrame(parentFrame : ModelFrame) : void
getModel() : Model
setModel(model : Model) : void
getDisplay() : String
setDisplay(display : String) : void
getInput() : DataElementSet
setInput(des : DataElementSet) : void
getOutput() : DataElementSet
setOutput(des : DataElementSet) : void
getKey() : String
getFrame() : JInternalFrame
getDesktop() : AnalysisDesktop
getChildren() : ModelFrame[]
doClose() : void
doOK() : void
doCancel() : void
doApply() : void
getGraphPanel() : JScrollPane
needsSeparatePanel(de : DataElement) : boolean
isEditable(de : DataElement) : boolean
isEditable(des : DataElementSet) : boolean
getIOPanel() : JComponent
getDESPanel(des : DataElementSet, isInput : boolean) : JComponent
getDEPanel(dataElement : DataElement, isInputDE : boolean) : JComponent
getSeparateDEPanel(dataElement : DataElement) : JComponent
-mFrame
JInternalFrame
(from swing)
-mInput
-mOutput
-mCopyInput
-mCopyOutput
-mParentFrame
LinkCanvas
(from asac)

The AnalysisDesktop Class

The AnalysisDesktop Class provides the interface for adding ModelFrames to the desktop. The AnalysisDesktop provides a hierarchical relationship between ModelFrames. A list of properties and methods for this class can be found in Table 6-22.

Table 6-22. Properties and Methods for AnalysisDesktop Class

Public Methods:	
AnalysisDesktop () :	Constructor.
getDesktop () : JdesktopPane	Returns the actual desktop.
getModelFrameCount () : int	Returns the number of frames on the desktop.
getModelFrameAt (I : int) : ModelFrame	Returns the frame at the specified index.
remove (modelFrame : Model-Frame) : void	Removes a frame from the desktop.
add (modelFrame : Model-Frame) : void	Adds a frame to the desktop.
getChildren (parentFrame : ModelFrame) : ModelFrame[]	Returns the children of a ModelFrame on the desktop.
getModelFrame (key : String) : ModelFrame	Returns the modelframe with the specified key.
addOrRaise (modelFrame : ModelFrame) : void	Adds or raises the frame on the desktop.
getAction (s : String) : Action	Returns the specified desktop action.

The ModelFrame Class

ModelFrames are the internal desktop frames on the AnalysisDesktop. Each ModelFrame has a parent (except the top level) and children, which are Model-Frames. The ModelFrames also may contain a set of input and output DataElementSets that may be edited or viewed by the user or updated by the server. A list of properties and methods for this class can be found in Table 6-23.

Table 6-23. Properties and Methods for DataElementSet Class

Public Methods:	
ModelFrame () :	Constructor.
getParentFrame () : ModelFrame	Returns the parent model frame.
setParentFrame (parentFrame : ModelFrame) : void	Specifies the parent model frame.
getModel () : Model	Returns the model.
setModel (model : Model) : void	Specifies the model.
getDisplay () : String	Returns the display .
setDisplay (display : String) : void	Specifies the display.
getInput () : DataElementSet	Returns the input DataElementSet.

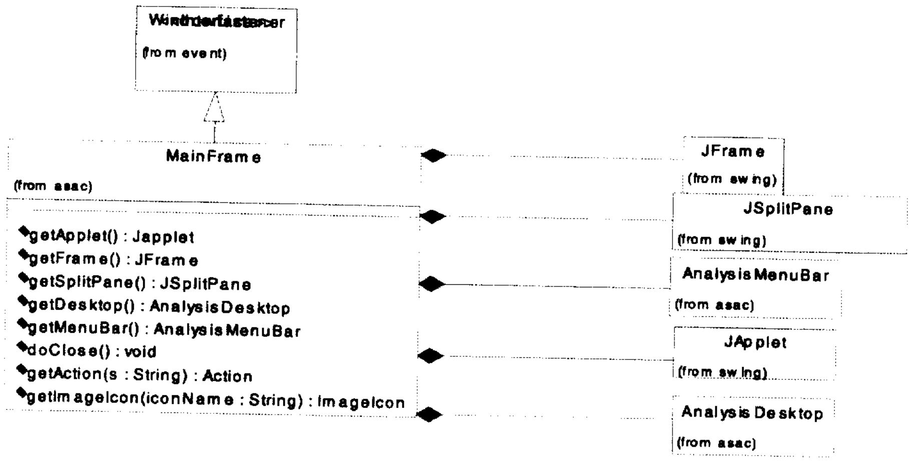
Table 6-22. Properties and Methods for AnalysisDesktop Class (Continued)

setInput (des : DataElementSet) : void	Specifies the input DataElementSet.
getOutput () : DataElementSet	Return the output DataElementSet.
setOutput (des : DataElementSet) : void	Specifies the output DataElementSet.
getKey () : String	Returns the unique key for the modelframe.
getFrame () : JinternalFrame	Returns the internal frame.
getDesktop () : AnalysisDesktop	Returns the desktop.
getChildren () : ModelFrame[]	Returns the children on the desktop.
doClose () : void	Closes the modelframe.
doOK () : void	Performs the OK button action.
doCancel () : void	Performs the Cancel button action.
doApply () : void	Performs the Apply button action.
getGraphPanel () : JScrollPane	Returns the graph panel or parent of the model-frame.
needsSeparatePanel (de : DataEle-ment) : boolean	Checks if the DataElement needs to be displayed on a separate panel.
isEditable (de : DataElement) : boolean	Checks if the DataElement is editable.
isEditable (des : DataElementSet) : boolean	Checks if the DataElementSet is editable.
getIOPanel () : Jcomponent	Returns the display panel for the input and output DataElementSets.
getDESPanel (des : DataElementSet, isInput : boolean) : Jcomponent	Returns the display panel for the DataElementSet.
getDEPanel (dataElement : DataElement, isInputDE : boolean) : Jcomponent	Returns the display panel for the DataElement.
getSeparateDEPanel (dataElement : DataElement) : Jcomponent	Returns the display panel for the DataElement when it needs a separate panel.

THE FRAME PACKAGE

The Frame Package contains the main window frame. The class diagram is shown in Figure 6-23.

Figure 6-23. Frame Package Diagram



The MainFrame Class

The MainFrame is the main window frame of the ASAC EA. A list of properties and methods for this class can be found in Table 6-24.

Table 6-24. Properties and Methods for MainFrame Class

Public Methods:	
getApplet () : Japplet	Returns the applet.
getFrame () : Jframe	Returns the frame.
getSplitPane () : JSplitPane	Returns the SplitPane object.
getDesktop () : AnalysisDesktop	Returns the desktop.
getMenuBar () : AnalysisMenu-Bar	Returns the MenuBar.
doClose () : void	Unregisters with the ORB and closes the mainframe.
getAction (s : String) : Action	Returns the specified action.
getImageIcon (iconName : String) : ImageIcon	Gets an image from local disk or URL.

DSSA Substage 4-7: Develop State Diagrams

The Client GUI uses colors and icons to display the states of Models and Links or breakpoints. The actual states of the objects are maintained by the AnalysisServer. Thus, the state diagrams for the client classes are the same as the state diagrams for the classes on the AnalysisServer. The Model class on the Client uses the same states as the DataTransformer class on the AnalysisServer. The Models have four states: Waiting, Running, Done and Error. The Client GUI uses an icon to represent each state:

- A Clock represents Waiting.
- A Flag represents Running.
- A Check represents Done.
- An "X" represents Error.

In addition to these icons, the border of the model's label on the analysis graph uses color to represent the state.

- Blue for Waiting
- Green for Running
- Green for Done
- Red for Error.

Each Analysis scenario is a subclass of Model and, therefore, inherits the states and icons from the Model class.

The Links or breakpoints are similar to the DataRelationship class on the AnalysisServer and, consequently, uses the same state diagrams. The Links have three states, Clear, Break Before, and Break After, to represent where the breakpoint will occur. The Links have an arrow drawn on the line to indicate direction of the data transformation. The Links use line color with an icon to represent the state.

- For a Clear Link, the line is blue.
- For a Break Before Link, the line is red with an icon near the starting Model.
- For a Break After Link, the line is red with an icon near the ending Model.

The Links also use a Boolean attribute to determine if the breakpoint has been reached.

DSSA Substage 4-8: Develop Deployment Diagrams

The Deployment Diagram is the same as shown in Figure 6-9.

DSSA Substage 4-9: Review and Iterate

Review and iterate the items developed in DSSA stage 4.

DSSA Stage 5—Identify Reusable Artifacts

The goal for this phase of the domain-engineering process is to populate the software architecture high-level design(s) with components that may be used to generate new applications in the domain.

The following substages of DSSA stage 5 will be completed during the ASAC design effort:

- 5-1 Develop and collect the reusable artifacts
- 5-2 Develop each module
- 5-3 Requirements, verification, and testing
- 5-4 Review and iterate.

DSSA Substage 5-1: Develop and Collect the Reusable Artifacts

No additional components need to populate the software architecture for the ASAC EA Beta version.

DSSA Substage 5-2: Develop Each Module

The development environment and process are the same as described in Chapter 5.

DSSA Substage 5-3: Requirements, Verification, and Testing

ASAC EA Beta Version Requirements

As previously mentioned, fifty-two requirements were applicable to the ASAC EA Beta version. They are:

Analysis Execution

- AE0001 The analyst shall have the capability to execute an analysis if an off-line administrator has granted the appropriate permissions.
- AE0002 The analyst shall have the capability to view and modify model input data at user-defined intermediate steps in the analysis. Any modifications to the model inputs shall be logged.
- AE0003 When an analysis is executed, the names of the models that are executed, as part of that analysis, will be logged to a log file.

- AE0004 When an analysis is executed, its inputs and outputs will be logged.
- AE0005 When a model is executed, its inputs and outputs will be logged.
- AE0006 Upon completion of the execution of an analysis, the results will be presented to the user if the user is logged into the system.
- AE0007 Analysis and Model outputs shall be viewable in both raw and converted format.
- AE0008 ASAC will provide a message to the user indicating a rough estimated time required to execute an analysis. Note: This will be a very rough estimate, as there are currently no plans to perform an interrogation of network and system(s) loading at the time of execution to provide a better estimate, not to mention the affect of data set size on model execution time.
- AE0009 ASAC EA shall support the execution of analyses in the "background" after users have logged off of the system.
- AE0010 The ASAC EA shall optionally mail a notification of analysis completion or suspension to the user, if the user is not logged into the system.
- AE0011 Users shall be able to cancel the execution of an analysis at any user pre-defined intermediate step.
- AE0012 Users shall be able to log back in and check the progress of, or cancel "active" analyses for which they have the appropriate permissions. When an analysis finishes, it shall remain "active" until the user views its outputs.
- AE0013 Analyses can be restarted from the beginning after their execution has finished or been canceled.
- AE0014 Users shall be able to set breakpoints on any data relationship. Breakpoints shall be settable before or after data conversion occurs in the data relationship.
- AE0015 Users shall be able to set preferences regarding e-mail delivery of various status messages that can get sent when they are not logged into the system.

Analysis Management

- AM0001 The capability shall be provided to create an analysis by using off-line tools.

- AM0002 The Analyst shall have the capability to view an existing analysis if an off-line administrator has granted the appropriate permissions.
- AM0003 The capability shall be provided to update an analysis by using off-line tools.
- AM0004 The Analyst shall have the capability to delete an analysis if an off-line administrator has granted the appropriate permissions.
- AM0005 The Analyst shall have the capability to copy an analysis if an off-line administrator has granted the appropriate permissions.
- AM0006 The capability shall be provided to store an analysis to the server for private or public use by using off-line tools.
- AM0007 The Analyst shall have the capability to store the results of an analysis to the server for private or public use if an off-line administrator has granted the appropriate permissions.

Analysis Specification

- AS0001 An analysis may contain one or more models or analyses.
- AS0002 Analyses may have default input values.
- AS0003 Default analysis input values may be overridden by the user.

Distributed Computing

- DC0001 ASAC will accommodate operation of its models at remote sites.
- DC0002 ASAC EA shall provide the capability to allow analysts to run more than one analysis concurrently.
- DC0003 ASAC EA shall support the concurrent execution of more than one instance of the same analysis on the same or different machines.
- DC0004 ASAC EA shall support the concurrent execution of more than one instance of the same model on the same or different machines.
- DC0005 The physical location of the models shall be transparent to the ASAC EA.
- DC0006 ASAC EA shall support a distributed application server model that allows multiple clients and servers to be located on different physical host machines.

- DC0007 ASAC EA shall allow users to run more than one analysis simultaneously.

Error Handling

- EH0001 The user shall be notified if the web server is not available (handled by the browser)
- EH0002 The user shall be notified if the analysis server is not available.
- EH0003 The user shall be notified if a model server is not available.
- EH0004 The user shall be notified if the analysis server encounters a failure during analysis execution.
- EH0005 The user shall be notified if a model server encounters a failure during model execution.
- EH0006 The user shall be notified if an invalid data type or value for analysis/model input is specified.
- EH0007 The user shall be notified if the database is not available or if a database access error is encountered.

General

- GE0001 The user application will have an intuitive graphical user interface that adheres to the IBM CUA standards.

Model Specification

- MS0001 Models shall have valid default values upon initialization (when added to an analysis).
- MS0002 An off-line administrator shall have the capability to add new models to the system by:

  Developing (or adding developed) models that match a well-defined interface.

  Creating model specifications in a TBD database that specifies the model parameters. e.g. inputs, outputs, and description.

  Writing and adding model wrappers that translate/map the well-defined model interface data element sets (DESs) to the model-specific interface for the model being added to the system. (i.e. translators from DESs to model inputs and translators from model outputs to DESs)

- MS0003 Models may have default input values.
- MS0004 Default Model input values may be overridden by the user.
- MS0005 EA model inputs may be an ASCII file.

Security

- SE0001 An off-line system administrator will define the level of authorization for analyses and scenarios on a per-user or per-group basis.
- SE0002 The owning user shall have permissions to view & execute an analysis if an off-line administrator has granted the appropriate permissions.
- SE0005 An off-line administrator shall control user access to models.
- SE0006 Users must log into the system.
- SE0007 User authentication must be at least as secure as HTTP basic authentication.
- SE0010 An off-line administrator shall be able to define groups of users for authorization. Users can belong to multiple groups.
- SE0011 Scenarios will have Read, Write, and Delete permissions associated with them. Analyses will only have Read permissions. Anybody able to read an analysis can create a scenario for that analysis and execute it.

These fifty-two requirements will be validated as part of the ASAC EA Beta version acceptance.

ASAC EA Beta Version Implementation

The analyses used for implementing the Beta version are the analyses that were implemented for the ASAC Model Integration Prototype (First Generation ASAC) and documented in the *Aviation System Analysis Capability Executive Assistant Design*.

ASAC EA Beta Version Testing

Procedures are being created for testing each of the ASAC EA Beta version requirements. The documentation and results of these tests will be published in the NASA Contractor Report for the ASAC EA for fiscal year 1999.

ASAC EA BETA VERSION RELEASE 1

The first release of the ASAC EA Beta version will be available for use by selected users on 31 October 1998. It will be accessible from the ASAC Web site at http://www.asac.lmi.org.

DSSA Substage 5-4: Review and Iterate

Review and iterate the items developed in DSSA stage 5.

Chapter 7
Conclusion

The work performed this fiscal year on the ASAC EA system builds upon the work documented in the *ASAC EA Architecture Description* and *the Aviation System Analysis Capability Executive Assistant Design.*

We successfully developed, tested, and demonstrated the ASAC EA POC to NASA in February and March 1998. We then used published and respected methodologies for expanding the ASAC EA POC design for the ASAC EA Beta version system. The expanded design includes a Use Case diagram, Interaction (Sequence and Collaboration) diagrams, Package diagrams, Class diagrams, State diagrams, and Deployment diagrams. We are completing the development of the ASAC EA Beta version system and plan to field the version in late October 1998.

We also evaluated OOD management systems for use in the ASAC EA system. Furthermore, we selected additional software libraries and development tools.

Work will continue on the ASAC EA Beta version, and the ASAC EA version 1.0 will be fielded in fiscal year 1999.

BIBLIOGRAPHY

Bellin, David and Susan Suchman Simone. "The CRC Card Book," Addison-Wesley, 1997.

Booch, Grady. "Object Solutions, Managing the Object-Oriented Project," Addison-Wesley, 1997.

Coad, Peter, and Mark Mayfield, "Java Design: Building Better Apps & Applets", Yourdon Press, 1997.

Common Object Request Broker Architecture, OMG, July, 1995.

Common Object Services Specification, OMG, March, 1995.

CORBAServices: Common Object Services Specification, Vol. 1, March 1995.

Domain Specific Software Architectures (DSSA), http://www.sei.cmu.edu/arpa/evo/dssa-sum.html.

Flanagan, David, "Java in a Nutshell", O'Reilly, 1997.

Fowler, Martin, and Kendall Scott. "UML Distilled—Applying the Standard Object Modeling Language," Addison-Wesley, 1997.

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. "Design Patterns—Elements of Reusable Object-Oriented Software," Addison-Wesley, 1995.

Hughes, Cameron, & Hughes, Tracey. Object-Oriented Multithreaded Using C++. John Wiley & Sons, Inc. New York. 1997.

Lockheed Martin Advanced Concepts Center and Rational Software Corporation. "Succeeding with the Booch and OMT Methods, A Practical Approach," Addison Wesley, 1996.

McConnell, Steve, "Code Complete", Microsoft Press, 1993.

Mowbray, Thomas J. and Ron Zahavi. "The Essential CORBA," Wiley, 1995.

Orfali, Robert, and Dan Harkey, "Client/Server Programming with Java and CORBA", Wiley Computing Publishing, 1997.

Orfali, Robert, Dan Harkey, and Jeri Edwards. "Instant CORBA," Wiley, 1997.

Orfali, Robert, Dan Harkey, and Jeri Edwards. "The Essential Client/Server Survival Guide," Wiley, 1996.

Orfali, Robert, Harkey, Dan, and Jeri Edwards. "The Essential Distributed Objects Survival Guide," Wiley, 1996.

Quatrani, Terry, "Visual Modeling with Ratoinal Rose and UML", Addison-Welsey, 1998.

Rational Software Corporation UML Resource Center, "UML Document Set Version 1.1," September 1997, http://www.rational.com/uml/references/.

Roberts, Eileen, and James A. Villani. "ASAC Executive Assistant Architecture Description Summary," NASA Contractor Report 201681, April 1997.

Roberts, Eileen, James A. Villani, Mohammed Osman, David Godso, Brent King, and Michael Ricciardi. "Aviation System Analysis Capability Executive Assistant Design," NASA Contractor Report 207679, May 1998.

Rumbaugh, James, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. "Object-Oriented Modeling and Design," Prentice Hall, 1991.

Tracz, W. and L. Coglianese. "Domain-Specific Software Architecture Engineering Process Guidelines, ADAGE-IBM-92-02B." Loral Federal Systems, Owego, 1992.

Appendix A
ASAC EA POC As-Run Test Procedures

This appendix contains the six procedures that were completed during ASAC EA POC testing. They are

- Analysis Execution test procedures TP-AE-1 and TP-AE-2
- Analysis Management test procedure TP-AM-1
- Analysis Specification procedure TP-AS-1
- Distributed Computing procedure TP-DC-1
- Error Handling procedure TP-EH-3.

Test Procedure TP-AE-1

Requirements tested:

- **AE0001** - The Analyst shall have the capability to execute an analysis if an off-line administrator has granted the appropriate permissions.
- **AE0006** - Upon completion of the execution of an analysis, the results will be presented to the user.
- **AE0008** - ASAC will provide a message to the user indicating a rough estimated time required to execute an analysis.

Operator Actions	Expected Results	Pass or Fail and PR Number
1. Login to riker and change directory to `"/home/kander/ea_poc"`.	1. None	—
2. Run the Visibroker SmartAgent in the background. Type: `"osagent &"`	2. None.	—
3. Run the ModelServer. Type: `"ModelServer -configFile poc.cfg &"`	3. The ModelServer starts. `"CORBA Server Running..."` is displayed.	pass
4. Run the AnalysisClient. Type: `"AnalysisClient -analysis poc -logLevel 2"`	4. Analysis is started, and a time estimate is displayed.	pass
5. Wait for the analysis to finish.	5. Analysis finishes and displays results: `"asm = 160824.742268 profit = -5172.724227"`	pass

TEST PROCEDURE TP-AE-2

Requirements tested:

- **AE0003** - When an analysis is executed, the names of the models that are executed, as part of that analysis, will be logged to a log file.
- **AE0004** - When an analysis is executed, both its default and user-defined inputs and outputs will be logged to a log file.
- **AE0005** - When a model is executed, both its default and user-defined inputs and outputs will be logged to a log file.

Operator Actions	Expected Results	Pass or Fail and PR Number
1. Login to riker and change directory (cd) to "`/home/kander/ea_poc`".	1. None	—
2. Run the Visibroker SmartAgent in the background. Type: "`osagent &`"	2. None.	—
3. Run the ModelServer. Type: "`ModelServer -configFile poc.cfg &`"	3. The ModelServer starts. "`CORBA Server Running...`" is displayed.	pass
4. Run the AnalysisClient. Type: "`AnalysisClient -analysis poc -logFile log -logLevel 5`"	4. The analysis is run and its results are displayed. Results should be: "`asm = 160824.742268 profit = -5172.724227`"	pass
5. Display the contents of the log file. Type: "`more log`".	5. Log file is displayed, containing, in this order, the names of the models created, the links (data relationships) between the models, the analysis input, the input & output for each model, and the analysis output.	pass

TEST PROCEDURE TP-AM-1

Requirements tested:

- **AM0001** - The capability shall be provided to create an analysis by using off-line tools.
- **AM0003** - The capability shall be provided to update an analysis by using off-line tools.

Operator Actions	Expected Results	Pass or Fail and PR Number
1. Login to riker and change directory to "`/home/kander/ea_poc`".	1. None	
2. Run the Visibroker SmartAgent in the background. Type: "`osagent &`"	2. None.	
3. Run the ModelServer. Type: "`ModelServer -configFile poc.cfg &`"	3. The ModelServer starts. "`CORBA Server Running...`" is displayed.	pass
4. Inspect the file "`poc.as`" and confirm that it exists and contains the specification for an analysis (i.e. that it has been created). Type "`more poc.as`".	4. The contents of the Analysis specification file are displayed. It contains, a description, a time estimate, a list of data transformers, a list of data relationships, a set of inputs, and a set of outputs.	Pass, PR 1
5. Run the AnalysisClient. Type: "`AnalysisClient -analysis poc`"	5. The analysis is run and its results are displayed. Results should be: "`asm = 160824.742268` `profit = -5172.724227`"	pass
6. Edit (update) the file "`poc.as`" using vi, emacs, or similar tool. Change the "passengers" variable in the "inputs" section from 156 to 300, and save the changes.	6. The Analysis specification is updated.	pass

TEST PROCEDURE TP-AS-1

Requirements tested:

- **AS0001** - An analysis may contain one or more models or analyses.
- **AS0002** - Analyses may have default input values.

Operator Actions	Expected Results	Pass or Fail and PR Number
1. Login to riker and change directory to "`/home/kander/ea_poc`".	1. None	—
2. Run the Visibroker SmartAgent in the background. Type: "`osagent &`".	2. None.	—
3. Run the ModelServer. Type: "`ModelServer -configFile poc.cfg &`".	3. The ModelServer starts. "`CORBA Server Running...`" is displayed.	pass
4. Examine the specification for the analysis named as1. Type "`more as1.as`".	4. The "dataTransformers" section of the analysis contains two models (traffic & cost), and two transformers which are analyses (revenue-a & profit-a). Also, the "inputs" section lists 5 variables, two that are "WAITING" (require user input), the others are "READY" (have defaults).	pass
5. Confirm that revenue-a & profit-a are really analyses. List all analysis specifications by typing "`ls *.as`".	5. A list of all analysis specifications are displayed, including revenue-a & profit-a.	pass
6. Run the AnalysisClient. Type: "`AnalysisClient -analysis as1`".	6. User is prompted for 2 inputs which do not have default values. The other inputs (which have default values) are not prompted for.	pass
7. Enter values for passengers and stage_length when prompted. Enter "300" for passengers and "1000" for stage_length.	7. The analysis runs, and results are displayed: Results should be: "`asm = 309278.350515` `profit = -9938.085052`"	Pass

TEST PROCEDURE TP-DC-1

Requirements tested:

- **DC0001** - ASAC will accommodate operation of its models at remote sites.
- **DC0003** - ASAC EA shall support the concurrent execution of more than one instance of the same analysis on the same or different machines for one or more users.
- **DC0004** - ASAC EA shall support the concurrent execution of more than one instance of the same model on the same or different machines for one or more users.
- **DC0005** - The physical location of the models shall be transparent to the ASAC EA.

Operator Actions	Expected Results	Pass or Fail and PR Number
1. Login to riker and change directory to "`/home/kander/ea_poc`".	1. None.	—
2. Run the Visibroker SmartAgent in the background. Type: "`osagent &`"	2. None.	—
3. Run the ModelServer. Type: "`ModelServer -configFile riker.cfg -logLevel 4 &`"	3. The ModelServer starts. "`CORBA Server Running...`" and other messages are displayed.	pass
4. Login to worf and change directory to "`/home/kander/ea_poc`".	4. None.	—
5. Run the ModelServer. Type: "`ModelServer -configFile worf.cfg -logLevel 4 &`"	5. The ModelServer starts. "`CORBA Server Running...`" and other messages are displayed.	pass
6. Login to spock and change directory to "`/home/kander/ea_poc`".	6. None.	—
7. Run the ModelServer. Type: "`ModelServer -configFile spock.cfg -logLevel 4 &`"	7. The ModelServer starts. "`CORBA Server Running...`" and other messages are displayed.	pass
8. Verify the models are running on each machine. Type "`osfind`" on riker.	8. A list of models is displayed, showing traffic & profit models running on riker, cost & revenue models running on worf, and traffic & revenue models running on spock.	pass
9. Open two additional windows on riker & run the AnalysisClient in two different windows simultaneously. Type: "`AnalysisClient -analysis poc`" in each window.	9. Analyses are run, identical results are displayed.	pass
10. Examine the output from the ModelServers to ensure that requirement DC0004 was met.	10. Each model of the same name should have run more or less simultaneously, some on the same machine, some on different machines.	pass
11. Open an additional window on worf & run the AnalysisClient on riker and worf simultaneously. Type: "`AnalysisClient -analysis poc`" on each machine.	11. Analyses are run, identical results are displayed.	pass
12. Again, examine the output from the ModelServers to ensure that requirement DC0004 was met.	12. Same as #10 above.	pass

TEST PROCEDURE TP-EH-3

Requirements tested:

- **EH0003** - The user shall be notified if a model server is not available.

Operator Actions	**Expected Results**	**Pass or Fail and PR Number**
1. Login to riker and change directory to `"/home/kander/ea_poc"`.	1. None.	—
2. Run the Visibroker SmartAgent in the background. Type: `"osagent &"`	2. None.	—
3. Run the ModelServer. Type: `"ModelServer -configFile eh3.cfg &"`	3. The ModelServer starts. `"CORBA Server Running..."` is displayed.	pass
4. Verify the profit model is not running. Type `"osfind"` on riker.	4. A list of object names (models) is displayed. The profit model should not be among them.	pass
5. Run the AnalysisClient. Type: `"AnalysisClient -analysis poc"`	5. Analysis is run, and an error should be generated that the profit model could not be found. Should display: `"Model Server Unavailable: profit"`	pass

Appendix B
Abbreviations

ASAC	Aviation System Analysis Capability
AST	Advanced Subsonic Technology program
BOA	basic object adapter
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
CRC	Class-Responsibility-Collaboration
DBMS	Database Management System
DES	DataElementSet
DSSA	domain-specific software architecture
EA	Executive Assistant
FAA	Federal Aviation Administration
GUI	graphical user interface
IDL	Interface Definition Language
MB	megabyte
NASA	National Aeronautics and Space Administration
OMG	Object Management Group
OMT	object modeling technique
OO	object oriented
OOD	object-oriented design
ORB	object request broker
PERL	Practical Extraction and Report Language
POC	Proof of Concept
QRS	Quick Response System
RAM	random-access memory
RCS	revision control system
TBD	to be determined
UML	Unified Modeling Language

WWW	**World Wide Web**

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1999	3. REPORT TYPE AND DATES COVERED Contractor Report

4. TITLE AND SUBTITLE Aviation System Analysis Capability Executive Assistant Development	5. FUNDING NUMBERS C NAS2-14361 WU 538-16-11-01
6. AUTHOR(S) Eileen Roberts, James A. Villani, Kevin Anderson, and Paul Book	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Logistics Management Institute 2000 Corporate Ridge McLean, Virginia 22102-7805	8. PERFORMING ORGANIZATION REPORT NUMBER NS801S1
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Langley Research Center Hampton, VA 23681-0001	10. SPONSORING / MONITORING AGENCY REPORT NUMBER NASA/CR-1999-209119

11. SUPPLEMENTARY NOTES

Langley Technical Monitor: Robert E. Yackovetsky
Final Report

12a. DISTRIBUTION / AVAILABILITY STATEMENT Unclassified - Unlimited Subject Category 01 Availability: NASA CASI (301) 621-0390 Distribution: Nonstandard	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

In this technical document, we describe the development of the Aviation System Analysis Capability (ASAC) Executive Assistant (EA) Proof of Concept (POC) and Beta version. We describe the genesis and role of the ASAC system, discuss the objectives of the ASAC system and provide an overview of components and models in the ASAC system, and describe the design process and the results of the ASAC EA POC and Beta system development. We also describe the evaluation process and results for applicable COTS software. The document has seven chapters, a bibliography, and two appendices.

14. SUBJECT TERMS ASAC, NASA, Development, Executive Assistant	15. NUMBER OF PAGES 160
	16. PRICE CODE A08

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Unlimited